# Non-Interactive Publicly-Verifiable Delegation of Committed Programs

Riddhi Ghosal [*] Amit Sahai [†] Brent Waters [‡]

**Abstract.** In this work, we present the first construction of a fully non-interactive publicly-verifiable delegation scheme for committed programs. More specifically, we consider a setting where Alice is a trusted author who delegates to an untrusted worker the task of hosting a program $P$, represented as a Boolean circuit. Alice also commits to a succinct value based on $P$. Any arbitrary user/verifier *without knowledge of* $P$ should be convinced that they are receiving from the worker an actual computation of Alice's program on a given input $x$.
Before our work, the only object known to imply this challenging form of delegation was a SNARG/SNARK for $\mathcal{NP}$. This is because from the point of view of the user/verifier, the program $P$ is an unknown witness to the computation. However, constructing a SNARG for $\mathcal{NP}$ from standard assumptions remains a major open problem.
In our work, we show how to achieve delegation in this challenging context assuming only the hardness of the Learning With Errors (LWE) assumption, bypassing the apparent need for a SNARG for $\mathcal{NP}$.

## 1 Introduction

We consider a scenario where a trusted software author Alice wishes to make it possible for a set of users to make use of her program $P$, which we treat as a (non-uniform) Boolean circuit. In particular, this program $P$ may have embedded within it a large proprietary database that Alice's program makes use of. However, Alice neither wants to release her program $P$ nor does she want to host and execute the program herself. Instead she wishes to delegate this computation to an untrusted Worker, and the User/Verifier wants to be certain that they are receiving an output obtained via a computation of Alice's actual program $P$. As illustrated in Figure 1, the way this works is:

1. Alice sends the program $P$ along with some computed $\mathsf{state}$ to the Worker, and Alice also publishes a succinct hash $H_P$ of her program, which the User/Verifier obtains. This step is done once and for all.
2. An Input Provider chooses an input $x$, which is sent to both the Worker and the User/Verifier. Note that the input provider could be some public source of information like a news channel of bulletin board, and need not involve the User/Verifier.

[*]UCLA, riddhi@cs.ucla.edu
[†]UCLA, sahai@cs.ucla.edu
[‡]UT Austin and NTT Research, bwaters@cs.utexas.edu

3. Finally, the Worker computes the output $y = P(x)$ along with a succinct proof $\Pi$, and sends both of these to the User/Verifier. Steps 2 and 3 may be repeated polynomially many times.

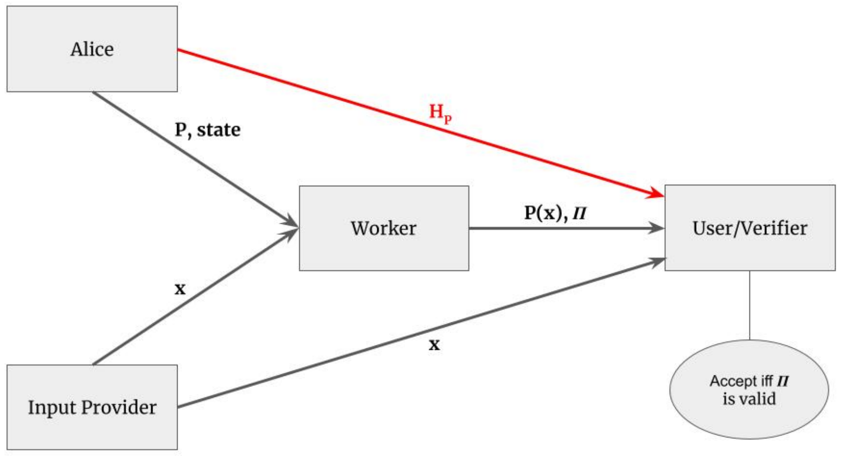

Fig. 1: The Delegation Setup

As illustrated in Figure 1, this process involves no back–and–forth communication. The communication is entirely unidirectional – which we call non-interactive – from left to right. Furthermore, we say that this scenario is *succinct* if all communication to the User/Verifier, and the runtime of the User/Verifier, is $\mathsf{poly}(\log |P|, \lambda, |x|)$, where $\lambda$ is a security parameter.

*Remark 1.* Note that on one hand, the Worker is trusted with the program $P$ by Alice, whereas, it is not trusted by the verifier. This asymmetry of trust is inherent in our setup and is well motivated. In a typical real world situation, the verifier is typically a user on the internet who takes part in a one off interaction with a cloud service for some computation. The need to prove honesty in this situation is significant. On the other hand, Alice might be able to have an agreement with the cloud service before handing over her program, which would make it hard for their Worker to breach trust without consequences.

*Comparison to Prior Work.* What we have just described is one of the most challenging variants of the classical problem of *publicly verifiable delegation* which has been the subject of intense work for decades, for many relaxed variations of the model that we describe above.

Specifically, delegation schemes *without public verification* based on standard assumptions for deterministic and non-deterministic computations have been designed [40,12,38,37,11,22,39,25,1,6,7,24]. Restricting verification to a *designated* verifier implies that the worker needs to produce a fresh proof unique for each particular verifier for any computation, which is certainly not ideal. Another line of work [15] achieves public verification but does not achieve public delegation. In other words, the input provider needs to run a pre-processing algorithm corresponding to the program $P$ before being able to delegate. Another model which has been extensively explored is when the User/Verifier is allowed to have interaction with the Worker, i.e., *interactive delegation.* Influenced by the first work on interactive efficient arguments by Kilian [27], there have been several works from standard assumptions [24,33,34,5] and some even unconditional soundness[17,36]. These are however not applicable in our setting where only one-way communication is permitted between the parties, as can be seen in the acyclic graph in Figure 1.

With regard to *non-interactive publicly verifiable delegation*, Starting from the seminal work on computationally sound proofs by Micali [31] in the random oracle model, there have been several constructions on publicly verifiable non-interactive delegation schemes [2,3,13,4,16,18,28,32] based on the *Random Oracle Model* or non-standard *knowledge assumptions.* From more standard assumptions, there have been several works recently [1,6,7,23]. An illustrative example is the recent work of [23] that proposed the first publicly verifiable non-interactive delegation scheme from a falsifiable decisional assumption on groups with bilinear pairings. However, in contrast with the setting we describe above, they can only achieve succinct delegation when the Verifier *knows the program $P$*. In our setting of Boolean circuits, this trivializes the delegation problem, since reading $P$'s description takes as long as evaluating $P$. Indeed, the case that we consider — where Alice's program is large — is extremely well motivated: the program $P$ could be an ML model with billions of painstakingly learned parameters.

*The SNARGs for $\mathcal{NP}$ barrier.* Why has constructing a protocol that caters to the fully non-interactive setting which we have defined been so elusive? Note that in our problem, the User/Verifier and Input Provider do not know the program $P$. Hence, from User/Verifier's perspective, $P$ is an $\mathcal{NP}$ witness. Thus, it certainly seems that finding a solution is intricately related to a major goal in the area of non interactive succinct proof systems, i.e., *SNARGs for $\mathcal{NP}$*. Unfortunately, the only known constructions of SNARGs for $\mathcal{NP}$ base their soundness on the *Random Oracle Model* or non-standard *knowledge assumptions.* Finding a solution solely relying on standard assumptions has been an open problem for over a decade. In fact, the closest that we have come is the very recent work achieving SNARGs for $\mathcal{P}$ [10] (see also [26]).

The major technical contribution in our work is to enable *Non-Interactive Publicly Verifiable Succinct Delegation for Committed Programs* without having to use SNARGs for $\mathcal{NP}$.

*Our Contribution:* We present the first complete solution to achieving succinct non interactive publicly verifiable delegation for committed programs. Indeed, furthermore, we can also achieve zero-knowledge guarantees as well. Our only computational assumption is the hardness of the *Learning with Errors* (LWE) problem. Somewhat surprisingly, we show that SNARGs for $\mathcal{NP}$ are not required to solve this problem, even though the statement being proved looks like an $\mathcal{NP}$ statement to the Verifier!

Instead, we show that many ideas from SNARGs for $\mathcal{P}$ [10] can in fact be applied here. Although $P$ is unknown to the User/Verifier, we show that it suffices for Alice to communicate a tiny amount of information of size $\mathsf{poly}(\log|P|)$ about the program $P$ (referred to as $H_P$) as shown in Figure 1. Because Alice is the author of $P$, this $H_P$ can be *trusted* as correctly generated. We stress that Alice does not need to know $x$ to compute $H_P$, hence this achieves public delegation and public verification in the completely non-interactive model described above. This leads to our main theorem,

**Theorem 1.** *Assuming the hardness of the LWE problem, Figure 2 gives a construction for publicly verifiable non-interactive succinct delegation for committed programs with CRS size, proof size and verifier time* $\mathsf{poly}(\lambda, \log|P|, |x|)$ *and prover run time being* $\mathsf{poly}(\lambda, |P|)$.

Finally, in order to get zero-knowledge, it suffices for Alice to commit to $H_P$ rather than sending it out in the open. We then present a generic transformation to convert any delegation protocol of this form to attain zero-knowledge.

**Theorem 2.** *Assuming the hardness of the LWE problem and existence of a succinct delegation scheme, Figure 5 gives a construction for publicly verifiable succinct delegation scheme with zero knowledge such that CRS size, proof size and verifier time are* $\mathsf{poly}(\lambda, \log|P|)$ *and prover run time is* $\mathsf{poly}(\lambda, |P|)$.

Finally, we also show how to achieve *zero knowledge* versions of our delegation scheme, meeting the same strong succinctness and efficiency goals, and under the same assumption (LWE).

We present a more detailed explanation in the Technical Overview.

## 2 Technical Overview

*Our Delegation Scenario* Let us briefly recall the setup of our delegation scenario. There are 4 parties, namely, (1) Alice-the program author $\mathsf{ProgAuth}$ who sends a program $P$ and some computed state $\mathsf{state}$ to a Worker, (2) an Input Provider $I$ that outputs some value $x$, (3) Worker $W$ that takes as input $(P, \mathsf{state}, x)$ and outputs $P(x)$ and a proof $\Pi$, and (4) User/Verifier $V$ gets as inputs $(x, P(x), \Pi)$ and outputs 1 if and only if $\Pi$ was a valid proof. Assume that all the parties get the security parameter $\lambda$ as an input. An additional requirement is that $|\Pi|$ and runtime of $V$ is $\mathsf{poly}(\lambda, \log|P|, |x|)$, and $W$ runs in time $\mathsf{poly}(\lambda, |x|, |P|)$. Thus, any non-interactive publicly verifiable succinct delegation scheme can be viewed as a collection of 4 algorithms: $\mathsf{sDel} = (\mathsf{ProgAuth}, W, I, V)$ with the input output behaviour and efficiency guarantees as specified. Note that this is indeed a $\mathcal{P}$ computation for the Worker but the primary challenge is that the verifier does not have knowledge of the "witness" $P$, hence this is an $\mathcal{NP}$ computation from the verifier's point of view. In this work, we observe that it is indeed feasible to achieve our delegation scenario for all circuits without having to go through SNARGs for $\mathcal{NP}$. Our technique is based on the recent work of Choudhuri et. al. [10] on SNARGs for $\mathcal{P}$. We begin by giving a brief overview of their approach and elaborate the challenges of directly incorporating their methodology for our setting.

*Challenges of implementing [10]* Roughly, the work of [10] uses Batch Arguments for $\mathcal{NP}$ (BARGs), which they build from LWE. BARGs allow an efficient prover to compute a non-interactive and publicly verifiable "batch proof" of many $\mathcal{NP}$ instances, with size $\mathsf{poly}(|w|\log T)$ for $T$-many $\mathcal{NP}$ statements with each witness of size $|w|$. They begin by looking at $P$ as a Turing machine and the steps of $P$'s computation are interpreted as an *Index Circuit* $C_{\mathsf{index}}$. Say, $P$ terminates in $T$ steps. Formally, they construct a BARG for the *Index Language* $\mathcal{L}^{\mathsf{index}}$, where

$$\mathcal{L}^{\mathsf{index}} = \{(C_{\mathsf{index}}, i) | \exists w_i, \text{ such that } C(i, w_i) = 1\},$$

where $i \in [T]$ is an index. Let $s_0, s_1, \ldots, s_T$ denote the encoding of internal states of $P$ along with its tape information, and let *Step* be its step function such that $\mathit{Step}(s_{i-1}) = s_i$ The witness for the $i^{th}$ intermediate computation is then defined as $w_i = (s_{i-1}, s_i)$. The index circuit is built such that $(C_{\mathsf{index}}, i) \in \mathcal{L}^{\mathsf{index}}$ essentially implies that the Turing machine step function was correctly computed on $s_{i-1}$ to yield $s_i$. Note that this alone does not suffice as a proof because the BARG only confirms that $(s_{i-1}, s_i)$ and $(s'_i, s_{i+1})$ are valid witnesses. If $s_{i-1}, s_i, s'_i, s_{i+1}$ are generated by the step function of the same Turing machine $P$, they must be consistent with each other, i.e., $s_i = s'_i$. However, this is not guaranteed by a BARG.

To resolve this issue, the prover also sends a *Somewhere Extractable Hash (SE)* to the witnesses $(s_0, \{s_{i-1}, s_i\}_{i\in[T]})$. The extraction property of this hash allows the verifier to check if the witness of two consecutive BARG instances are indeed consistent with each other. At this stage, we would like to remind the reader of their efficiency goals where crucially, they desire proof size and verification time to be $\mathsf{poly}(\lambda, \log T)$. However, note that $|C_{\mathsf{index}}|$ grows linearly with $|s_i|$ and the known constructions [20] of SE hashes can only produce hashes with size $\mathsf{poly}(|s_i|)$. This means that total communication and verifier run time will be at least $\mathsf{poly}(|s_i|)$. This is certainly no good if the Turing machine has massive states. To overcome this final barrier, they make use of Hash Trees which compress the states $s_i$ to a short hash $h_i$ such that $|h_i| = \mathsf{poly}(\lambda)$. Such trees [30] also have a soundness property where a Prover must produce a succinct proof $\Pi_i$ that the hash tree was indeed implemented correctly at the $i^{th}$ step of the Turing machine computation. Once the succinctness guarantee is ensured, the prover then produces SE hashes corresponding to $(h_0, \Pi_0, \{h_{i-1}, \Pi_{i-1}, h_i, \Pi_i\}_{i\in[T]})$ along with the openings to these hashes. To summarise, the proof consists of two parts, (1) The BARG proof, and (2) A *somewhere extractable* hash of the witnesses. Relying on the soundness of BARG, extraction correctness property of SE hash and soundness of the Hash Tree, a User/Verifier can check if each of these $T$ intermediate steps are indeed the correct states for $P$, i.e., the computation was done honestly.

However, this approach only works if User/Verifier can confirm that the inputs used for the computation by the Worker, i.e. $(P, x)$ are indeed the correct starting values as provided by the Program Author and Input Provider. This works fine for [10] because in their setting, the User/Verifier actually knows $(P, x)$. Unfortunately, this is not at all true in our scenario. Thus, the techniques of Choudhuri et al. [10] cannot be implemented directly as the soundness of the BARG proof cannot provide any guarantees if there is no way for to check that the initial inputs used by the Worker are correct.

*Our Idea.* We start with an alternate way of interpreting the computation of $P$ on input $x$ as the following: Consider a Circuit-Universal Turing Machine $\mathcal{TM}$ which takes as input $P, x, y$ and accepts $(P, x, y)$ in $T = \tilde{O}(|P|)$ steps if $P(x) = y$. We can assume without loss of generality that $P \in \{0,1\}^m$, $x \in \{0,1\}^n$ and $y \in \{0,1\}$, where $m, n \leq 2^\lambda$. Keeping this in mind, we introduce the notion of *Semi-Trusted SNARGs* for $\mathcal{NP}$. This new kind of SNARG is one that will work for general $\mathcal{NP}$ computations, but only with a little bit of extra help from a trusted party that knows the witness – which in our delegation scenario is Alice, who knows the witness $P$!

A Semi-Trusted SNARG is a tuple of algorithms: $\mathsf{stSNARG} = (\mathsf{Setup}, \mathsf{TrustHash}, \mathsf{P}, \mathsf{V})$, where (1) $\mathsf{Setup}$ is a randomised algorithm that takes as input the security parameter and outputs a Common Random String (CRS). (2) a *trusted* deterministic $\mathsf{TrustHash}$ takes as input the (CRS, $P$) and outputs a digest $H_P$, (3) a deterministic prover $\mathsf{P}$ which takes as input CRS and $(P, x, y)$, and outputs a proof $\Pi$, and (4) a deterministic verifier $\mathsf{V}$ which gets CRS,$(H_P, x, y, \Pi)$ as input and outputs 1 iff $\Pi$ is valid. It must be that $|\Pi|$ and run time of $\mathsf{V}$ is $\mathsf{poly}(\lambda, \log T)$, and $\mathsf{P}$ runs in time $\mathsf{poly}(\lambda, |x|, |P|, T)$. A simple reduction shows that in the CRS model (or alternatively in a model where Alice chooses the CRS), existence of $\mathsf{stSNARG}$ implies the existence of $\mathsf{sDel}$. We show this formally in Lemma 11. Hence, from here onwards, our goal is to construct a *Semi-Trusted SNARG for* $\mathcal{NP}$.

We briefly provide an informal explanation of our construction.

Like [10], every intermediate state of the Universal Turing Machine is encoded into a succinct hash (call it $\mathsf{h}_0, \ldots, \mathsf{h}_T$) accompanied with succinct proofs $\{\Pi_i\}_{i \in [T]}$. The prover computes two independent copies of *Somewhere Extractable* (SE) hashes $(c_1, c_2)$ of the encoding $\{\mathsf{h}_0, \{(\mathsf{h}_1, \Pi_1), \ldots, (\mathsf{h}_T, \Pi_T)\}\}$ along with their corresponding openings. Here $\mathsf{h}_0 = (\mathsf{st}_0, H_P, H_x, H_{work})$, where $\mathsf{st}_0$ is that hash of $\mathcal{TM}$'s starting state which is publicly known, $H_x$ denote the hash of $x$, and $H_{work}$ is the hash of $\mathcal{TM}$'s blank work tape. The use of two independent SE hashes are pivotal for soundness which we elaborate later.

We point out that $\mathsf{TrustHash}$ computes $H_P$ using the same hash tree which is used for hashing the Turing machine states by the Prover. This is crucial to ensure soundness of the protocol. We show in Figure 3 that once the public hash is fixed by $\mathsf{TrustHash}$, one can hard code $(y, c_1, c_2, T, H_P, H_x)$ to the index circuit

$C_{\mathsf{index}}$ for BARG. At this point, we can now follow the approach from [10]. $\mathsf{V}$ can rely upon the binding property/collision resistance of the hash to ensure that the prover has used $P$ and $x$ which were provided by Alice and the input provider respectively. The main observation here is that once a trusted party fixed a hash of the program $P$ and $V$ is convinced that computation was commenced with the correct inputs, the soundness of BARG, extraction correctness of the SE hash and soundness of hash tree ensures that the semi-trusted SNARG construction is sound.

While our proof of soundness closely follows the blueprint of [10], we choose to present our proof in a different, and arguably simpler, way. In [10], *No-Signaling Somewhere Extractable*(NSSE) hashes are used extensively. In our proof, we choose to omit explicit use of this notion, and instead we make direct use of two independent SE hashes as mentioned above. A simple hybrid argument then gives a straightforward proof for soundness. This shows that the "anchor and step" use of SE hashes, which dates to the introduction of somewhere-binding hashes [20] in 2015, is directly sufficient for this proof of soundness.

*Zero-Knowledge* We have only discussed soundness guarantees thus far. However, in our delegation scenario, it might also be extremely important to ensure that no information about $P$ leaked to $\mathsf{V}$ during the delegation process. Hence it is important to add *zero-knowledge* guarantees to our protocol. We finally give a generic transformation to modify a semi-trusted SNARG to add zero knowledge guarantees. In order to do so we make use of a statistically binding extractable commitment scheme and a NIZK [1], and roughly make the following modifications:

- We add an additional commitment to 0 in the CRS which is never used in the proof but helps in proving zero knowledge.
- The public hash output by $\mathsf{TrustHash}$ is a binding commitment $C_P$ of $H_P$. It then sends $(P, H_P)$ to the worker $W$ only.
- The SE hashes $c_1, c_2$ are also committed as a part of the proof and not published in the open.
- The prover wraps the BARG $\Pi$ with a NIZK which proves that that the BARG verification circuit indeed accepts the BARG proof.
- The Verifier then checks if the NIZK proof is valid.

The binding and hiding property of the commitment, and *witness indistinguishability* of NIZK guarantees zero knowledge.

## 3 Preliminaries

We use some standard tools as building blocks to perform the Succinct Delegation.

[1]Multi-thoerem NIZK from LWE is possible by combining [35] and [14]. Note that the weaker notion NIWI would also suffice to achieve zero knowledge in our setting.

- **Somewhere Extractable Hash [20,10,9]:**
  $\mathsf{SE} = (\mathsf{SE.Gen}, \mathsf{SE.TGen}, \mathsf{SE.Hash}, \mathsf{SE.Open}, \mathsf{SE.Verify}, \mathsf{SE.Ext})$
- **Non Interactive Batch Arguments** ($\mathsf{BARG}$) **for Index Language[10]:**
  $\mathsf{BARG} = (\mathsf{BARG.Gen}, \mathsf{BARG.TGen}, \mathsf{BARG.Prove}, \mathsf{BARG.Verify})$
- **Hash Tree[23,30]:**
  $\mathsf{HT} = \mathsf{HT.Gen}, \mathsf{HT.Hash}, \mathsf{HT.Read}, \mathsf{HT.Write}, \mathsf{HT.VerRead}, \mathsf{HT.VerWrite}$
- **Non Interactive Zero Knowledge Argument[35,8,19]:**
  $(\mathsf{NIZK} = \mathsf{NIZK.Gen}, \mathsf{NIZK.Prove}, \mathsf{NIZK.V})$
- **Statistically Binding Extractable Commitment[29]:**
  $\mathsf{Com}_{\mathsf{bind}} = (\mathsf{Com.Gen}, \mathsf{Com.TGen}, \mathsf{Com.C}, \mathsf{Com.Ext})$

We use all the primitives in a standard way as prior works. The hash tree can be constructed from any collision resistant hash function. The others are known to be instantiated from LWE. Formal definitions and properties of the primitives can be found in the Supplementary Material.

## 4 Publicly Verifiable Non Interactive Succinct Delegation

We formally define the notion of Publicly Verifiable Non Interactive Succinct Delegation ($\mathsf{sDel}$) which is similar to the definition proposed in prior works [21]. Such a delegation scheme in the CRS model involves the following PPT algorithms, (1)Software/Program Author $\mathsf{ProgAuth}$ (3)Cloud Worker $W$ , and (3) Verifier $V$An $\mathsf{sDel}$ comprises of the following polynomial time algorithms:

- $\mathsf{sDel.Setup}(1^\lambda)$: A randomized setup algorithm which on input security parameter $\lambda$ and outputs $\mathsf{crs}$.
- $\mathsf{sDel.ProgAuth}(1^\lambda, \mathsf{crs})$: A program author which takes as input $\lambda$, outputs a (not public) program $P \in \{0,1\}^m$, $m \leq 2^\lambda \in \mathbb{N}$, $\mathsf{state}$ and a public digest $H_P$.
- $\mathsf{sDel}.W(\mathsf{crs}, P, \mathsf{state}, H_P, x)$: A deterministic cloud worker which on input $\mathsf{crs}$, program $P$, input $x \in \{0,1\}^n, n \leq 2^\lambda \in \mathbb{N}$ outputs a value $y$ and proof $\Pi$.
- $\mathsf{sDel}.V(\mathsf{crs}, x, y, H_P, \Pi)$: A deterministic verifier which on input $\mathsf{crs}$, digest $H_P, x, y, \Pi$ either accepts or rejects.

A publicly verifiable succinct delegation scheme $(\mathsf{sDel.Setup}, \mathsf{sDel.ProgAuth}, \mathsf{sDel}.W, \mathsf{sDel}.V)$ satisfies the following properties:

- **Completeness.** For every PPT program generating algorithm $\mathsf{sDel.ProgAuth}$, every $\lambda, n, m \in \mathbb{N}$, and for all $x \in \{0,1\}^n$ such that $n, m < 2^\lambda$, we have

$$\begin{aligned}\Pr[\mathsf{sDel}.V(\mathsf{crs}, x, y, H_P, \Pi) = 1 \wedge P(x) = y \big| \mathsf{crs} \leftarrow \mathsf{sDel.Setup}(1^\lambda),\\ ((P, \mathsf{state}), H_P) \leftarrow \mathsf{sDel.ProgAuth}(1^\lambda, \mathsf{crs}),\\ (y, \Pi) \leftarrow \mathsf{sDel}.W(\mathsf{crs}, P, \mathsf{state}, H_P, x)] = 1.\end{aligned}$$

- **Efficiency.** $\mathsf{sDel.Setup}$ runs in time $\mathsf{poly}(\lambda)$, $\mathsf{sDel}.W$ runs in time $\mathsf{poly}(\lambda, |P|, |x|)$ and outputs a proofs of length $\mathsf{poly}(\lambda, \log |P|, |x|)$, and $\mathsf{sDel}.V$ runs in time $\mathsf{poly}(\lambda, \log |P|, |x|)$.

- **Soundness.** For every PPT adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2)$, every PPT program generating algorithm $\mathsf{sDel.ProgAuth}$, and the tuple $n = n(\lambda), m = m(\lambda)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$,

$$\begin{aligned}\Pr[\mathsf{sDel}.V(\mathsf{crs}, x, y, H_P, \Pi) = 1 \wedge P(x) \neq y \big| , \mathsf{crs} \leftarrow \mathsf{sDel.Setup}(1^\lambda),\\ ((P, \mathsf{state}), H_P) \leftarrow \mathsf{sDel.ProgAuth}(1^\lambda, \mathsf{crs}),\\ (x, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs}), (y, \Pi) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{state}, H_P, x, \mathsf{aux})]\\ \leq \mathsf{negl}(\lambda).\end{aligned}$$

To construct sDel, we introduce a notion of *Semi-Trusted Succinct Non-Interactive Arguments* stSNARG which we formally introduce and construct in Section 5. After that, we prove the following lemma (cf. Lemma 11) which shows how to construct sDel using stSNARG as a building block.

**Lemma 1.** *Assuming $T = \mathsf{poly}(m, n)$, $T, m, n \leq 2^\lambda$, the* stSNARG *protocol in Figure 2 implies the unconditional existence of a publicly verifiable non interactive succinct delegation scheme* sDel *as defined above.*

## 4.1 sDel with Zero-Knowledge

A publicly verifiable non interactive succinct delegation scheme with zero knowledge $\mathsf{zk-sDel}$ is defined by the following efficient algorithms:

- $\mathsf{zk-sDel.Setup}(1^\lambda)$: A randomized setup algorithm which on input security parameter $\lambda$ and outputs $\mathsf{crs}$.
- $\mathsf{zk-sDel.ProgAuth}(1^\lambda, \mathsf{crs})$: A program author which takes as input $\lambda$, generates a program $P \in \{0,1\}^m$, $m \leq 2^\lambda \in \mathbb{N}$. Additionally, it computes a digest $H_P$ and creates a statistically binding and extractable commitment $C_P$ of $H_P$ under randomness $r$. Finally it sends a private output $(P, \mathsf{state})$ and public output $C_P$. Here $\mathsf{state}$ contains the randomness $r$ and $H_P$ encoded in it along with any other state information.
- $\mathsf{zk-sDel}.W(\mathsf{crs}, P, \mathsf{state}, C_P, x)$: A deterministic cloud worker which on input $\mathsf{crs}$, program $P$, commitment $C_P$, $x \in \{0,1\}^n, n \leq 2^\lambda \in \mathbb{N}$ outputs a value $y$ and proof $\Pi$.
- $\mathsf{zk-sDel}.V(\mathsf{crs}, x, y, C_P, \Pi)$: A deterministic verifier which on input $(\mathsf{crs}, C_P, x, y, \Pi)$ either accepts or rejects.

Apart from the Completeness, Efficiency and Soundness guarantees mentioned above, a publicly verifiable succinct delegation scheme
$(\mathsf{zk-sDel.Setup}, \mathsf{zk-sDel.ProgAuth}, \mathsf{zk-sDel}.W, \mathsf{zk-sDel}.V)$ satisfies the following additional property:

**Non Interactive Zero Knowledge.** For all $\lambda, n, m \in \mathbb{N}$ such that $n, m \leq 2^\lambda, \forall, x \in \{0,1\}^n$ and $y \in \{0,1\}$, there exists a PPT simulator $\mathsf{Sim} := (\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3)$ such that the distributions of

$$(\mathsf{crs}, x, y, C_P, \mathit{\Pi}) \Big| (\mathsf{crs}, \mathsf{aux}) \leftarrow \mathsf{Sim}_1(1^\lambda), (C_P, \mathsf{aux}') \leftarrow \mathsf{Sim}_2(\mathsf{crs}, \mathsf{aux}), \\ (y, \mathit{\Pi}) \leftarrow \mathsf{Sim}_3(\mathsf{aux}', \mathsf{crs}, x, C_P)$$

and

$$(\mathsf{crs}, x, y, C_P, \mathit{\Pi}) \Big| \mathsf{crs} \leftarrow \mathsf{zk} - \mathsf{sDel.Setup}(1^\lambda), ((P, \mathsf{state}), C_P) \leftarrow \mathsf{zk} - \mathsf{sDel.ProgAuth}(1^\lambda, \mathsf{crs}), \\ (y := P(x), \mathit{\Pi}) \leftarrow \mathsf{zk} - \mathsf{sDel}.W(\mathsf{crs}, P, \mathsf{state}, x, C_P)$$

are indistinguishable.

In Section 6, we present a generic construction of a semi trusted non-interactive succinct arguments with zero-knowledge (ZKstSNARG) from stSNARG. Analogous to the previous lemma, we get the following corollary(cf. Corollary 2) from Lemma 11

**Corollary 1.** *Assuming $T = \mathsf{poly}(m, n)$, $T, m, n \leq 2^\lambda$, the* ZKstSNARG *protocol in Figure 5 implies the unconditional existence of a publicly verifiable non interactive succinct delegation scheme with zero knowledge.*

## 5 Semi-Trusted Succinct Non-Interactive Argument (stSNARG)

We introduce a notion of "Semi-Trusted" SNARGs which is similar to the general definition of SNARGs with an addition "trusted" polynomial time algorithm that outputs a hash for the witness. Further, we provide an explicit construction of an stSNARG for all of $NP$. Note that any SNARG for arbitrary $NP$ language $\mathcal{L}$ can be reformulated as a Turing Machine which takes in as input an instance $x$ along with witness $w$ and accepts $x, w$ in $T$ steps if $x \in \mathcal{L}$ [10]. In this work, we modify the definition of [10] by using a Universal Turing Machine $\mathcal{TM}$ which takes as input an instance $(x, y)$, a witness which is a program $P$ and accepts $(P, x, y)$ in $T$ steps if $P(x) = y$. We formalise this notion as follows:

Let $\mathcal{TM}$ be a Universal Turing Machine which takes as input a program $P \in \{0,1\}^m$ for some $m < 2^\lambda$, and $x \in \{0,1\}^n$ for some $n < 2^\lambda$ and $y \in \{0,1\}$ which serve as an input and output for $P$ respectively. $\mathcal{TM}$ accepts $(P, x, y)$ in $T$ steps if $P(x) = y$. A prover produces a proof $\mathit{\Pi}$ to convince a verifier that $\mathcal{TM}$ accepts $P, x, y$ in $T$. A publicly verifiable semi-trusted SNARG (stSNARG) for $\mathcal{TM}$ has the following polynomial time algorithms:

- $\mathsf{stSNARG.Setup}(1^\lambda, 1^T)$: A randomized setup algorithm which on input security parameter $\lambda$, and number of Turing Machine steps $T$, outputs $\mathsf{crs}$.
- $\mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$: A deterministic and honest algorithm which on input $\mathsf{crs}$ and a program $P \in \{0,1\}^m$ for some $m < 2^\lambda$, outputs a succinct and public digest $H_P$ of $P$ corresponding to $\mathsf{crs}$.
- $\mathsf{stSNARG.P}(\mathsf{crs}, P, x, y, H_P)$: A deterministic prover algorithm which on input the $\mathsf{crs}$, $P \in \{0,1\}^m$ for some $m < 2^\lambda$, $x \in \{0,1\}^n$ for some $n < 2^\lambda$, $y \in \{0,1\}$ and the digest $H_P$ outputs a proof $\mathit{\Pi}$.

– $\mathsf{stSNARG.V}(\mathsf{crs}, x, y, H_P, \varPi)$: A deterministic verification algorithm which on input $\mathsf{crs}$, $x$, $y$, digest $H_P$ and proof $\varPi$, either accepts(output 1) or rejects(output 0) it.

A Universal Turing Machine $\mathcal{TM}$ on input $(P, x, y)$ outputs 1 if it accepts $(P, x, y)$ within $T$ steps. We define the $NP$ language $\mathcal{L}_{\mathcal{TM}}$ as,

$$\mathcal{L}_{\mathcal{TM}} \coloneqq \{(P, x, y, T, H_P, \mathsf{crs}) \big| \mathcal{TM}(P, x, y) = 1 \wedge \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P) = H_P\}.$$

Note that here $P$ is not considered a part of the witness although it is unknown to the verifier because a typical NP statement puts a there exists constraint on the witness. In that case, the statement becomes trivial because there will always exist a program $P$ which on input $x$ ignores the input and outputs $y$. We need to ensure that $P$ is the program output by the program author independent of $x$. Moreover, this is indeed a $\mathsf{P}$ statement for the prover.

A publicly verifiable stSNARG scheme $\mathsf{stSNARG} =$ $(\mathsf{stSNARG.Setup}, \mathsf{stSNARG.TrustHash}, \mathsf{stSNARG.P}, \mathsf{stSNARG.V})$ satisfies the following properties:

– **Completeness.** For every $\lambda, T, n, m \in \mathbb{N}$ such that $T, n, m < 2^\lambda$, program $P \in \{0,1\}^m$, input $x \in \{0,1\}^n$ and output $y \in \{0,1\}$ such that $(P, x, y, T, H_P, \mathsf{crs}) \in \mathcal{L}_{\mathcal{TM}}$, we have

$$\Pr[\mathsf{stSNARG.V}(\mathsf{crs}, x, y, H_P, \varPi) = 1 \big| \mathsf{crs} \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T),$$
$$H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P), \varPi \leftarrow \mathsf{stSNARG.P}(\mathsf{crs}, P, x, y, H_P)] = 1.$$

– **Efficiency.** $\mathsf{stSNARG.Setup}$ runs in time $\mathsf{poly}(\lambda, T)$, $\mathsf{stSNARG.TrustHash}$ runs in time $\mathsf{poly}(\lambda, |P|, T)$, $\mathsf{stSNARG.P}$ runs in time $\mathsf{poly}(\lambda, |x|, |P|, T)$ and outputs a proofs of length $\mathsf{poly}(\lambda, \log T)$, and $\mathsf{stSNARG.V}$ runs in time $\mathsf{poly}(\lambda, \log T)$.

– **Soundness.** For every PPT adversary $\mathcal{A} \coloneqq (\mathcal{A}_1, \mathcal{A}_2)$ and the tuple $T = T(\lambda), n = n(\lambda), m = m(\lambda)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{stSNARG.V}(\mathsf{crs}, x, y, H_P, \varPi) = 1 \wedge (P, x, y, T, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \big|,$$
$$\mathsf{crs} \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T), (P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs}),$$
$$H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P), (x, y, \varPi) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, H_P, \mathsf{aux})] \leq \mathsf{negl}(\lambda).$$

**Protocol 1 (Semi-Trusted SNARG)**

```
– $\mathsf{stSNARG.Setup}(1^\lambda, 1^T)$ :
    • $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.Gen}(1^\lambda, 1^{M_{\lambda,T}}, 1^{L_\lambda})$[a]
    • $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.Gen}(1^\lambda, 1^M, 1^L)$
    • $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|})$
    • $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
    • *return* $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
– $\mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
    • $(\mathsf{tree}_0^2, \mathsf{rt}_0^2) \leftarrow \mathsf{HT.Hash}(\mathsf{dk}, P),\ H_P \leftarrow \mathsf{rt}_0^2$
    • *return* $H_P$.
– $\mathsf{stSNARG.P}(\mathsf{crs}, P, x, y, H_P)$ :
    • $\square :=$ *empty string*
    • $(\mathsf{tree}_0^1, \mathsf{rt}_0^1) \leftarrow \mathsf{HT.Hash}(\mathsf{dk}, x),\ (\mathsf{tree}_0^2, \mathsf{rt}_0^2) \leftarrow \mathsf{HT.Hash}(\mathsf{dk}, P),$
      $(\mathsf{tree}_0^3, \mathsf{rt}_0^3) \leftarrow \mathsf{HT.Hash}(\mathsf{dk}, \square)$
    • *initialize* $\mathsf{s}$ *with the* $\mathsf{start}$ *state of* $\mathcal{TM}$
    • $\mathsf{st}_0 := (0, 0, 0, \mathsf{s})$
    • $\mathsf{h}_0 := (\mathsf{st}_0, \mathsf{rt}_0^1, \mathsf{rt}_0^2, \mathsf{rt}_0^3)$
    • *for every* $i = 1$ *to* $T$,
          $\mathsf{rt}_i^1 \leftarrow \mathsf{rt}_{i-1}^1, \mathsf{rt}_i^2 \leftarrow \mathsf{rt}_{i-1}^2$
          $(l_i^1, l_i^2, l_i^3) \leftarrow \mathsf{StepR}(\mathsf{st}_{i-1})$
          $\left\{(b_i^j, \Pi_i^j) \leftarrow \mathsf{HT.Read}(\mathsf{tree}_{i-1}^j, l_i^j)\right\}_{j\in[3]}$
          $(b_i', l_i', \mathsf{st}_i) \leftarrow \mathsf{StepW}(\mathsf{st}_{i-1}, b_i^1, b_i^2, b_i^3)$
          $(\mathsf{tree}_i^3, \mathsf{rt}_i^3, \Pi_i') \leftarrow \mathsf{HT.Write}(\mathsf{tree}_{i-1}^3, l_i'^3, b_i'^3)$
          $\mathsf{h}_i \leftarrow (st_i, \mathsf{rt}_i^1, \mathsf{rt}_i^2, \mathsf{rt}_i^3)$
    • $A := \left(\mathsf{h}_0, \left(\mathsf{h}_1, \{b_1^j, \Pi_1^j\}_{j\in[3]}, \Pi_1'\right), \dots, \left(\mathsf{h}_T, \{b_T^j, \Pi_T^j\}_{j\in[3]}, \Pi_T'\right)\right)$
    • $c_{\mathsf{even}} \leftarrow \mathsf{SE.Hash}\,(\mathsf{SE}.K_{\mathsf{even}}, A)$ *and* $c_{\mathsf{odd}} \leftarrow \mathsf{SE.Hash}\,(\mathsf{SE}.K_{\mathsf{odd}}, A)$
    • $c := (c_{\mathsf{even}}, c_{\mathsf{odd}})$
    • $I_x \leftarrow \{[i_1, i_2] \big| A[i_1, i_2] = x\}$
    • $\rho_{\mathsf{h}_0} \leftarrow \mathsf{SE.Open}(\mathsf{SE}.K_{\mathsf{even}}, A, I_{\mathsf{h}_0})$[b]
    • *for every* $i \le [\lfloor T/2 \rfloor]$,
          *for* $B \in \{\mathsf{h}_{2i}, \{b_{2i}^j, \Pi_{2i}^j\}_{j\in[3]}, \Pi_{2i}'\},\ \rho_B \leftarrow \mathsf{SE.Open}(\mathsf{SE}.K_{\mathsf{even}}, A, I_B)$
    • *for every* $i \le [\lfloor T/2 \rfloor]$,
          *for* $B \in \{\mathsf{h}_{2i+1}, \{b_{2i+1}^j, \Pi_{2i+1}^j\}_{j\in[3]}, \Pi_{2i+1}'\},\ \rho_B :=$
          $\mathsf{SE.Open}(\mathsf{SE}.K_{\mathsf{odd}}, A, I_B)$
    • *Let* $C_{\mathsf{index}}$ *be as defined in Figure 3*
    • $\Pi := \mathsf{BARG.P}\Big(\mathsf{crs}, C_{\mathsf{index}}, \mathsf{h}_0, \{\mathsf{h}_{i-1}, \mathsf{h}_i, \{b_i^j, \Pi_i^j\}_{j\in[3]}, \Pi_i', \rho_{\mathsf{h}_{i-1}}, \rho_{\mathsf{h}_i},$
      $\{\rho_{b_i^j}, \rho_{\Pi_i^j}\}_{j\in[3]}, \rho_{\Pi_i'}\}_{i\in[T]}\Big)$
    • *return* $(c, \Pi)$ [c].
– $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi))$ :
    • *Compute* $C_{\mathsf{index}}$
    • *return* 1 *if and only if* $\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1$.
```

[a] $M_{\lambda,T} = O(T\,\mathsf{poly}(\lambda))$ and $L_\lambda = O(\mathsf{poly}(\lambda))$ are arbitrary and efficiently computable values which can be fixed in advance and hardcoded to the Setup algorithm during instantiation.

[b] Note that for simplification, we abuse notation here by specifying opening to more than a single bit.

[c] We often abuse notation and use $(c, \Pi)$ to denote a proof. This can be done without loss of generalization by defining a new proof $\Pi' = (c \| \Pi)$.

Fig. 2: Semi-Trusted SNARG

**Circuit 1 (Circuit $C_{\mathsf{index}}$)**

```
– Hard-coded: $y, c, \mathsf{start}, \phi, \mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, T, H_P, H_x := \mathsf{HT.Hash}(\mathsf{dk}, x)$
– Input:
      $\big(i, (\mathsf{h}_i := (\mathsf{st}_i, \mathsf{rt}_i^1, \mathsf{rt}_i^2, \mathsf{rt}_i^3), \rho_{\mathsf{h}_i})\big)$, if $i = 0$
      $\Big(i, (\{\mathsf{h}_{i-1}, \mathsf{h}_i, \{b_i^j, \Pi_i^j\}_{j\in[3]}, \Pi_i', \rho_{\mathsf{h}_{i-1}}, \rho_{\mathsf{h}_i}, \{\rho_{b_i^j}, \rho_{\Pi_i^j}\}_{j\in[3]}, \rho_{\Pi_i'}\})\Big)$, $\forall i \in [T]$
– Output: return 1 if and only if
   • if $i = 0$
      a. $\mathsf{st}_0 = \mathsf{start}$
      b. $H_x = \mathsf{rt}_0^1$
      c. $H_P = \mathsf{rt}_0^2$
      d. $\mathsf{HT.Hash}(\mathsf{dk}, \square)$ has $\mathsf{rt}_0^3$ as root
   • else
      ∗ if $i$ is even:
         a. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{odd}}, c_{\mathsf{odd}}, \mathsf{h}_{i-1}, \rho_{\mathsf{h}_{i-1}}) = 1$
         b. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{even}}, c_{\mathsf{even}}, \mathsf{h}_i, \rho_{\mathsf{h}_i}) = 1$
         c. $\Big\{\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{even}}, c_{\mathsf{even}}, b_i^j, \rho_{b_i^j}) = 1\Big\}_{j\in[3]}$
         d. $\Big\{\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{even}}, c_{\mathsf{even}}, \Pi_i^j, \rho_{\Pi_i^j}) = 1\Big\}_{j\in[3]}$
         e. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{even}}, c_{\mathsf{even}}, \Pi_i', \rho_{\Pi_i'}) = 1$
      ∗ if $i$ is odd:
         a. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{even}}, c_{\mathsf{even}}, \mathsf{h}_{i-1}, \rho_{\mathsf{h}_{i-1}}) = 1$
         b. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{odd}}, c_{\mathsf{odd}}, \mathsf{h}_i, \rho_{\mathsf{h}_i}) = 1$
         c. $\Big\{\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{odd}}, c_{\mathsf{odd}}, b_i^j, \rho_{b_i^j}) = 1\Big\}_{j\in[3]}$
         d. $\Big\{\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{odd}}, c_{\mathsf{odd}}, \Pi_i^j, \rho_{\Pi_i^j}) = 1\Big\}_{j\in[3]}$
         e. $\mathsf{SE.Verify}(\mathsf{SE}.K_{\mathsf{odd}}, c_{\mathsf{odd}}, \Pi_i', \rho_{\Pi_i'}) = 1$
      ∗ $\phi(\mathsf{h}_{i-1}, \mathsf{h}_i, \{b_i^j, \Pi_i^j\}_{j\in[3]}, \Pi_i') = 1$
      ∗ if $i = T$
         a. $\mathsf{HT.Hash}(\mathsf{dk}, y)$ has $\mathsf{rt}_T^3$ as root.
         b. $\mathsf{st}_T$ indeed encodes the accept state.
```

Fig. 3: Circuit $C_{\mathsf{index}}$

## 5.1 Our Construction

Our construction is formulated similar to that of [10]. Specifically, we use the notion of non-interactive BARG for index language and SE Hash functions in our scheme.

*Setup for Universal Turing Machine.* For a cleaner analysis, we assume without loss of generality that $\mathcal{TM}$ consists of three tapes, namely, $\mathsf{Tp}_1, \mathsf{Tp}_2, \mathsf{Tp}_3$. $\mathsf{Tp}_1$ and $\mathsf{Tp}_2$ are read only tapes that store $x$ and $P$ respectively. $\mathsf{Tp}_3$ is the work tape which is initialized with $\square$ to denote an empty string.

*Transition steps for $\mathcal{TM}$.* $\mathcal{TM}$'s state information along with the head locations of the three tapes are encoded as $\mathsf{st}$. To handle Turing Machines with arbitrarily long tapes, we encode $\{\mathsf{Tp}_i\}_{i\in[3]}$ using three Hash Trees as defined in previous sections and produce tree roots $\mathsf{rt}^1, \mathsf{rt}^2, \mathsf{rt}^3$ respectively.

Let the each intermediate transition state of $\mathcal{TM}$ be encoded as $h_i :=$ $(\mathsf{st}_i, \mathsf{rt}_i^1, \mathsf{rt}_i^2, \mathsf{rt}_i^3)$ for $i \in [T]$. A single step of $\mathcal{TM}$ can be interpreted in the manner described below which is similar to one described for a RAM in [23]. We break down the step function at the $i^{th}$ stage into two deterministic polynomial time algorithms:

- $\mathsf{StepR}$: On input $\mathsf{st}_{i-1}$ of $\mathcal{TM}$, outputs head positions $l_{i-1}^1, l_{i-1}^2, l_{i-1}^3$ which denote the memory locations of $\mathsf{Tp}_1, \mathsf{Tp}_2, \mathsf{Tp}_3$ which $\mathcal{TM}$ in the current state $\mathsf{st}_{i-1}$ would read from.
- $\mathsf{StepW}$: On input $\mathsf{st}_{i-1}$, and bits $b_{i-1}^1, b_{i-1}^2, b_{i-1}^3$ outputs bit $b'$, location $l'$ and $\mathsf{st}_i$ such that $\mathcal{TM}$ upon reading $b_{i-1}^1, b_{i-1}^2, b_{i-1}^3$ at locations $l_{i-1}^1, l_{i-1}^2, l_{i-1}^3$ using $\mathsf{HT.Read}$, would write $b'$ at location $l'$ of $\mathsf{Tp}_3$, thereby transition to new state $\mathsf{st}_i$.

Now, we translate the $i^{th}$ single step of $\mathcal{TM}$ to the circuit $\phi$ which is defined such that on input digests $h_{i-1} := (\mathsf{st}_{i-1}, \mathsf{rt}_{i-1}^1, \mathsf{rt}_{i-1}^2, \mathsf{rt}_{i-1}^3)$ and $h_i :=$ $(\mathsf{st}_i, \mathsf{rt}_i^1, \mathsf{rt}_i^2, \mathsf{rt}_i^3)$, bits $b_i^1, b_i^2, b_i^3$, and proofs $\Pi_i^1, \Pi_i^2, \Pi_i^3, \Pi_i'$, $\phi(h_{i-1}, h_i, b_i^1, b_i^2, b_i^3, \Pi_i^1, \Pi_i^2, \Pi_i^3, \Pi_i') =$ 1 if and only if the following hold:

1. $(l_i^1, l_i^2, l_i^3) \leftarrow \mathsf{StepR}(\mathsf{st}_{i-1})$
2. $(b', l', \mathsf{st}') \leftarrow \mathsf{StepW}(\mathsf{st}_{i-1}, b_i^1, b_i^2, b_i^3)$
3. $\mathsf{st}' = \mathsf{st}_i$
4. $\mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}_{i-1}^1, l_i^1, b_i^1, \Pi_i^1) = 1$
5. $\mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}_{i-1}^2, l_i^2, b_i^2, \Pi_i^2) = 1$
6. $\mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}_{i-1}^3, l_i^3, b_i^3, \Pi_i^3) = 1$
7. $\mathsf{rt}_i^1 = \mathsf{rt}_{i-1}^1$
8. $\mathsf{rt}_i^2 = \mathsf{rt}_{i-1}^2$
9. $\mathsf{HT.VerWrite}(\mathsf{dk}, \mathsf{rt}_{i-1}^3, l', b', \mathsf{rt}_i^3, \Pi_i') = 1$

Here, $\mathsf{dk}$ denote the hash keys used to build the three hash trees. Note that the efficiency of hash tree implies that $\phi$ can be constructed such that it can represented as a formula in $L = \mathsf{poly}(\lambda)$ variables. For the $T$ steps of $\mathcal{TM}$, we have the following formula over $M = O(L \cdot T)$ variables:

$$\Phi(\mathsf{h}_0, \{h_i, b_i^1, b_i^2, b_i^3, \Pi_i^1, \Pi_i^2, \Pi_i^3, \Pi_i'\}_{i\in[T]}) = \bigwedge_{i\in[T]} \phi(h_{i-1}, h_i, b_i^1, b_i^2, b_i^3, \Pi_i^1, \Pi_i^2, \Pi_i^3, \Pi_i')$$

Following the techniques in [10], we use a combination of $\mathsf{SE}$ Hash along with $\phi$ to produce the circuit for index languages.

Our semi-trusted SNARG scheme is given in Figure 2 and the corresponding index language circuit is shown as Figure 3.

**Theorem 3.** *Assuming the existence of Somewhere Extractable Hash functions, non-interactive Batch Arguments for Index Languages, and Collision Resistant Hash Trees as described in section 3,, Figure 2 is a publicly verifiable non-interactive semi-trusted SNARG with CRS size, proof size and verifier time* $\mathsf{poly}(\lambda, \log T)$ *and prover run time being* $\mathsf{poly}(\lambda, T)$.

*Completeness.* Here we give a sketch arguing completeness of our scheme. Our construction in Figure 2 tells that

$$\begin{aligned}&\Pr[\mathsf{stSNARG.V}(\mathsf{crs}, x, y, H_P, \Pi) = 1 \big| \mathsf{crs} \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T),\\ &\quad H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P), \Pi \leftarrow \mathsf{stSNARG.P}(\mathsf{crs}, P, x, y, H_P)] =\\ &\qquad \Pr[\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1 \big| \mathsf{crs} \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T),\\ &\qquad\quad H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P), \Pi \leftarrow \mathsf{stSNARG.P}(\mathsf{crs}, P, x, y, H_P)]\end{aligned}$$

where $C_{\mathsf{index}}$ is the index circuit as shown in Figure 3. Observing $\mathsf{stSNARG.P}$ algorithm in our scheme tells it is sufficient to show that if the prover is honest and uses a valid witness, then $(C_{\mathsf{index}}, i) \in \mathcal{L}_{\mathsf{index}}, \forall i \in \{0\} \cup [T]$. If we can argue that this is indeed the case, then the completeness of $\mathsf{BARG}$ gives the desired result.

If $(P, x, y, T, H_P, \mathsf{crs}) \in \mathcal{L}_{\mathcal{TM}}$, then $(C_{\mathsf{index}}, 0) \in \mathcal{L}_{\mathsf{index}}$ is trivially true by observation. Now, let us look at $(C_{\mathsf{index}}, 1)$. We start by analysing that $\phi(\mathsf{h}_0, \mathsf{h}_1, \{b_1^j, \Pi_1^j\}_{j\in[3]}, \Pi_1') = 1$ is true. $\{\mathsf{rt}_1^i = \mathsf{rt}_0^i\}_{i\in[2]}$ follow from the read-only nature of tapes $\mathsf{Tp}_1, \mathsf{Tp}_2$. Since, $\left\{(b_1^j, \Pi_1^j) \leftarrow \mathsf{HT.Read}(\mathsf{tree}_0^j, l_1^j)\right\}_{j\in[3]}$, the hash tree completeness of read ensures that $\{\mathsf{HT.VerRead}(\mathsf{dk}, \mathsf{rt}_0^i, l_1^i, b_1^i, \Pi_1^i) = 1\}_{i\in[3]} = 1$ and $\{\mathsf{Tp}_i[l_1^i] = b_1^i\}_{i\in[3]}$. This along with the correctness of Turing Machine $\mathsf{StepR}$ function implies that $b_1^1, b_1^2, b_1^3$ are indeed the correct input for the $\mathsf{StepW}$ function of $\mathcal{TM}$. Finally, $(\mathsf{tree}_1^3, \mathsf{rt}_1^3, \Pi_1') \leftarrow \mathsf{HT.Write}(\mathsf{tree}_0^3, l_1', b_1')$ implies $\mathsf{HT.VerWrite}(\mathsf{dk}, \mathsf{rt}_0^3, l', b', \mathsf{rt}_1^3, \Pi_1') = 1$ from the hash tree completeness of write property. The same property also ensures that $\mathsf{Tp}_3$ changes only at the $l'^{th}$ memory location. When paired with the correctness of $\mathsf{StepW}$, we get that $\mathsf{st}_1 = \mathsf{st}'$

The completeness of the $\mathsf{SE}$ hash implies that the verification algorithm certainly accepts all the local openings. Thus, $(C_{\mathsf{index}}, 1) \in \mathcal{L}^{\mathsf{index}}$. Now, $(C_{\mathsf{index}}, T) \in \mathcal{L}^{\mathsf{index}}$ because $\mathcal{TM}$ accept $(P, x, y)$ in $T$ steps. We can show in a similar manner that for all other $i$, $(C_{\mathsf{index}}, i) \in \mathcal{L}^{\mathsf{index}}$. This proves the completeness of the scheme in Figure 2.

*Efficiency.*

- Runtime of $\mathsf{stSNARG.Setup}$ is $\mathsf{poly}(\lambda, T)$. This follows from the efficiency of underlying primitives.
- $\mathsf{stSNARG.TrustHash}$ computes $H_P$ in time $|P| \cdot \mathsf{poly}(\lambda)$ which is $\mathsf{poly}(|P|, \lambda)$.
- $|C_{\mathsf{index}}| = \mathsf{poly}(\lambda, \log T)$. This follows from the efficiency of the $\mathsf{SE}$ hash and the efficiency of hash tree construction.
- CRS Size: By the corresponding properties of the underlying primitives, $|\mathsf{crs}| = \mathsf{poly}(\lambda, \log T)$.
- The prover's computation time is dominated by the hashes corresponding to $x, P$ and the Turing Machine step functions that is run $T$ times. This requires a total time of $\mathsf{poly}(\lambda, |x|) + \mathsf{poly}(\lambda, |P|) + \mathsf{poly}(\lambda, T) = \mathsf{poly}(\lambda, |x|, |P|, T)$.
- Proof Length: $|c| + |\Pi| = \mathsf{poly}(\lambda, \log T) + \mathsf{poly}(\lambda, \log T, |C_{\mathsf{index}}|) = \mathsf{poly}(\lambda, \log T)$.
- Verifier Time: Time taken to compute $C_{\mathsf{index}}$ and verify the $\mathsf{BARG}$. This is $\mathsf{poly}(\lambda, \log T, |C_{\mathsf{index}}|) = \mathsf{poly}(\lambda, \log T)$.

*Soundness.* Let us assume for the sake of contradiction that our scheme in Figure 2 is not sound, i.e., there exists a PPT adversary $\mathcal{A} \coloneqq (\mathcal{A}_1, \mathcal{A}_2)$, a value $T$ and a polynomial function $\mathsf{poly}(\lambda)$ such that for infinitely many values of $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{G}^{\mathcal{A}} = 1] \geq \frac{1}{\mathsf{poly}(\lambda)},$$

where $\mathcal{A}$ plays Game $\mathsf{G}$ described below

```
Real Game $\mathsf{G}$
 1. $\mathsf{crs} \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T)$
 2. $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
 3. $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
 4. $((x, y)(c, \mathit{\Pi})) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, H_P, \mathsf{aux})$
 5. if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \mathit{\Pi})) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$,
    return 1
 6. else return 0
```

Let $S_i$ denote the following set:

$$S_i = \begin{cases} \mathsf{h}_0 & \text{if } i = 0 \\ \left\{\mathsf{h}_i, \{b_i\}_{j\in[3]}, \{\mathit{\Pi}_i\}_{j\in[3]}, \mathit{\Pi}'_i\right\} & \text{if } i \in [T] \end{cases}$$

Let $D$ denote the string $\Big(\mathsf{h}_0, \left\{\mathsf{h}_i, \{b_i\}_{j\in[3]}, \{\mathit{\Pi}_i\}_{j\in[3]}, \mathit{\Pi}'_i\right\}_{i\in[T]}\Big)$. $I_{S_i} \subset |D|$ denotes the following:

$$I_{S_i} = \left\{[a, b] \middle| a, b \in |D|, D[a, b] = S_i\right\}.$$

In game $\mathsf{G}$, say we have $(\mathsf{tree}^1_0, \mathsf{rt}^1_0) \leftarrow \mathsf{HT.Hash}(\mathsf{dk}, x)$, $(\mathsf{tree}^2_0, \mathsf{rt}^2_0) \leftarrow \mathsf{HT.Hash}(\mathsf{dk}, P)$, $(\mathsf{tree}^3_0, \mathsf{rt}^3_0) \leftarrow \mathsf{HT.Hash}(\mathsf{dk}, \square)$. Also, let $\mathsf{st}_0 \coloneqq (0, 0, 0, \mathsf{s})$, where $\mathsf{s}$ is the $\mathsf{start}$ state of $\mathcal{TM}$. We say that $\bar{\mathsf{h}}_0 \coloneqq (\mathsf{st}_0, \mathsf{rt}^1_0, \mathsf{rt}^2_0, \mathsf{rt}^3_0)$ defines a unique "true" digest for the starting step of $\mathcal{TM}$.

If $\mathsf{stSNARG.V}(\mathsf{crs}, x, H_P, c, \mathit{\Pi}) = 1$, then Algorithm $\mathit{Step}(x, P, \mathsf{crs}, i)$ in Figure 4 computes the unique true digest $\bar{\mathsf{h}}_i$ after the $i^{th}$ Turing Machine Step along with the other uniquely correct values of the set $\bar{S}_i \coloneqq \{\bar{\mathsf{h}}_i, \{\bar{b}_i\}_{j\in[3]}, \{\bar{\mathit{\Pi}}_i\}_{j\in[3]}, \bar{\mathit{\Pi}}'_i\}$. We use the notation $\mathit{Step}(x, P, \mathsf{crs}, i).x$ to denote $x \in \bar{S}_i$. We proceed by performing an induction on the following sequence outer hybrid games $\mathsf{G}_i, i$ from 1 to $T$. We use a sequence of inner hybrid games to transition between subsequent outer hybrids. Our induction hypothesis is that, under suitable assumptions, for all $i \in 1$ to $T$, there exists a negligible function $\lambda$ such that,

$$\Pr[\mathsf{G}^{\mathcal{A}} = 1] \leq \Pr[\mathsf{G}^{\mathcal{A}}_i = 1] + \mathsf{negl}(\lambda).$$

Intuitively, the $i^{th}$ game $G_i$ is similar to the real life soundness game with the following two changes: (1) The key generation for the $\mathsf{SE}$ hash and $\mathsf{BARG}$ is done in the trapdoor mode at the $i^{th}$ game. This allows for extractability of the $i^{th}$ block of the string $D$ from the commitment $c$. (2) The adversary wins the game if they break the soundness assumption as the real life game $\mathsf{G}$ and the extracted block is indeed the correct one.

```
Algorithm $Step(x, y, P, \mathsf{crs}, i)$

  – $\square :=$ empty string
  – $(\mathsf{tree}_0^1, \mathsf{rt}_0^1) := \mathsf{HT.Hash}(\mathsf{dk}, x)$, $(\mathsf{tree}_0^2, \mathsf{rt}_0^2) := \mathsf{HT.Hash}(\mathsf{dk}, P)$, $(\mathsf{tree}_0^3, \mathsf{rt}_0^3) :=$
    $\mathsf{HT.Hash}(\mathsf{dk}, \square)$
  – initialize $\mathsf{s}$ with the $\mathsf{start}$ state of $\mathcal{TM}$
  – $\mathsf{st}_0 := (0, 0, 0, \mathsf{s})$
  – $\bar{\mathsf{h}}_0 := (\mathsf{st}_0, \mathsf{rt}_0^1, \mathsf{rt}_0^2, \mathsf{rt}_0^3)$
  – if $i = 0$, return $\bar{S}_0 := (\mathsf{st}_0, \mathsf{rt}_0^1, \mathsf{rt}_0^2, \mathsf{rt}_0^3)$
  – else
      • for $\mathsf{count} = 1$ to $i$,
            $(l_{\mathsf{count}}^1, l_{\mathsf{count}}^2, l_{\mathsf{count}}^3) \leftarrow \mathsf{StepR}(\mathsf{st}_{\mathsf{count}-1})$
            $\{(b_{\mathsf{count}}^k, \Pi_{\mathsf{count}}^k) := \mathsf{HT.Read}(\mathsf{tree}_{\mathsf{count}-1}^k, l_{\mathsf{count}}^k)\}_{k \in [3]}$
            $(b_{\mathsf{count}}'^3, l_{\mathsf{count}}'^3, \mathsf{st}_{\mathsf{count}}) := \mathsf{StepW}(\mathsf{st}_{\mathsf{count}-1}, b_{\mathsf{count}}^1, b_{\mathsf{count}}^2, b_{\mathsf{count}}^3)$
            $(\mathsf{tree}_{\mathsf{count}}^3, \mathsf{rt}_{\mathsf{count}}^3, \Pi'_{\mathsf{count}}) := \mathsf{HT.Write}(\mathsf{tree}_{\mathsf{count}-1}^3, l_{\mathsf{count}}'^3, b_{\mathsf{count}}'^3)$
      • $\bar{\mathsf{h}}_i := (\mathsf{st}_i, \mathsf{rt}_i^1, \mathsf{rt}_i^2, \mathsf{rt}_i^3)$
      • $\bar{b}_i := (b_i^1, b_i^2, b_i^3)$
      • $\bar{l}_i := (l_i^1, l_i^2, l_i^3)$
      • $\bar{\mathsf{rt}}_i := \mathsf{rt}_{i-1}^1, \mathsf{rt}_{i-1}^2, \mathsf{rt}_i^3$
      • $\bar{\Pi}_i := (\Pi_i^1, \Pi_i^2, \Pi_i^3, \Pi'_i)$
      • return $\bar{S}_i := (\bar{\mathsf{h}}_i, \bar{b}_i, \bar{\mathsf{rt}}_i, \bar{\Pi}_i)$
```

Fig. 4: Turing Machine $i^{th}$ step.

```
Outer Hybrid Game $\mathsf{G}_i$
 1. if $i$ is even
        $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_i})$
        $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{i-1}})$
 2. if $i$ is odd
        $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_{i-1}})$
        $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_i})$
 3. $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, i)$
 4. $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
 5. $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
 6. $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
 7. $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
 8. $((x, y), (c, \Pi)) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
 9. Parse $c$ as $(c_{\mathsf{odd}}, c_{\mathsf{even}})$
10. if $i$ is even and $i \neq 0$
      • $(\mathsf{h}_i, \{b_i^k\}_{k \in [3]}, \{\Pi_i^k\}_{k \in [3]}, \Pi'_i) \leftarrow \mathsf{SE.Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$
      • $(\mathsf{h}_{i-1}, \{b_{i-1}^k\}_{k \in [3]}, \{\Pi_{i-1}^k\}_{k \in [3]}, \Pi'_{i-1}) \leftarrow \mathsf{SE.Ext}_{\mathsf{odd}}(c_{\mathsf{odd}}, \mathsf{SE}.K_{\mathsf{odd}})$
11. if $i$ is odd
      • $(\mathsf{h}_i, \{b_i^k\}_{k \in [3]}, \{\Pi_i^k\}_{k \in [3]}, \Pi'_i) \leftarrow \mathsf{SE.Ext}_{\mathsf{odd}}(c_{\mathsf{odd}}, \mathsf{SE}.K_{\mathsf{odd}})$
      • if $i-1 > 0$ then $(\mathsf{h}_{i-1}, \{b_{i-1}^k\}_{k \in [3]}, \{\Pi_{i-1}^k\}_{k \in [3]}, \Pi'_{i-1}) \leftarrow \mathsf{SE.Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$
      • if $i - 1 = 0$ then $\mathsf{h}_0 \leftarrow \mathsf{SE.Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$
12. if $\mathsf{stSNARG.V}\,(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \land ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \land \mathsf{h}_i =$
    $Step(x, y, P, \mathsf{crs}, i).\bar{\mathsf{h}}_i$, return 1
13. else return 0
```

*Base Case:* Assuming key indistinguishability and soundness of $\mathsf{SE}$ hash and $\mathsf{BARG}$, we need to show that $\Pr[\mathsf{G}^{\mathcal{A}} = 1] \le \Pr[\mathsf{G}_1^{\mathcal{A}} = 1] + \mathsf{negl}(\lambda)$.

We begin by using a sequence of hybrids to transition from $\mathsf{G}$ to an intermediate game $\mathsf{G}_0$. The colored texts in the hybrids below indicate the steps in the hybrids exclusively appear in a particular game. We only present proof sketches for the intermediate lemmas in this section due to lack of space. Concrete proofs have been shifted to the Supplementary Material.

```
Hybrid Games $\mathsf{G}_a, \mathsf{G}_b, \mathsf{G}_{ab}, \mathsf{G}_0$
 1. $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_0})$ ...($\mathsf{G}_a, \mathsf{G}_b, \mathsf{G}_{ab}, \mathsf{G}_0$)
 2. $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.Gen}(1^\lambda, 1^M)$ ...($\mathsf{G}_a$)
 3. $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_1})$ ...($\mathsf{G}_b, \mathsf{G}_{ab}, \mathsf{G}_0$)
 4. $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|})$...($\mathsf{G}_a, \mathsf{G}_b$)
 5. $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, 0)$...($\mathsf{G}_{ab}, \mathsf{G}_0$)
 6. $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
 7. $\mathsf{crs} := (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
 8. $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
 9. $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
10. if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$, return
    1
11. $\mathsf{h}_0 \leftarrow \mathsf{SE.Ext}_{\mathsf{even}}(c_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{even}})$...($\mathsf{G}_0$)...($\mathsf{G}_a, \mathsf{G}_b, \mathsf{G}_{ab}$)
12. if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_0 =$
    $Step(x, y, P, \mathsf{crs}, 0).\bar{\mathsf{h}}$, return 1 ...($\mathsf{G}_0$)
13. else return 0
```

**Lemma 2.** *Assuming key indistinguishability of* $\mathsf{SE}$, $\left|\Pr[\mathsf{G}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_a^{\mathcal{A}} = 1]\right| \le \mathsf{negl}(\lambda)$.

*Proof.* The only difference in Game $\mathsf{G}$ and $\mathsf{G}_a$ is that the key generation algorithm of the $\mathsf{SE}$ hash ($\mathsf{SE.Gen}$) is replaced by the trapdoor key generation ($\mathsf{SE.TGen}$).

If $\left|\Pr[\mathsf{G}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_a^{\mathcal{A}} = 1]\right| > \mathsf{negl}(\lambda)$, then one can construct a PPT adversary $\mathcal{B}$ that breaks the key indistinguishability of $\mathsf{SE}$ using $I_{S_0}$ with $\mathsf{Key}$ as input from the key generation algorithm of the $\mathsf{SE}$ hash and runs $\mathcal{A}$ on $\mathsf{Key}$. Here, $\mathsf{Key}$ is either $\mathsf{SE.Gen}(1^\lambda, 1^M)$ or $\mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_0})$ based on whether $\mathcal{A}$ is interacting with game $\mathsf{G}$ or $\mathsf{G}_a$ respectively. Note that the reduction can simulate the other steps of games $\mathsf{G}$ or $\mathsf{G}_a$. Now, the probability that $\mathcal{B}$ returns 1 in either case is exactly equal to the probability that $\mathcal{A}$ wins the corresponding games, hence, $\mathcal{B}$ breaks if $\left|\Pr[\mathsf{G}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_a^{\mathcal{A}} = 1]\right| \ge \mathsf{negl}(\lambda)$. This leads to a contradiction of our assumption.

**Lemma 3.** *Assuming key indistinguishability of* $\mathsf{SE}$, $\left|\Pr[\mathsf{G}_a^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_b^{\mathcal{A}} = 1]\right| \le \mathsf{negl}(\lambda)$.

This again follows from the key-indistinguishability of $\mathsf{SE}$ as shown in the previous lemma as the only difference in these games is that the key generation algorithm for the $\mathsf{SE}$ hash has been changed to $\mathsf{TGen}$, hence we skip the proof.

**Lemma 4.** *Assuming key indistinguishability of* $\mathsf{BARG}$, $\left|\Pr[\mathsf{G}_b^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_{ab}^{\mathcal{A}} = 1]\right| \le \mathsf{negl}(\lambda)$.

*Proof.* The only difference in Game $\mathsf{G}_b$ and $\mathsf{G}_{ab}$ is that the key generation algorithm of the BARG (BARG.Gen) is replaced by the trapdoor key generation (BARG.TGen) at index 0.

If $\left|\Pr[\mathsf{G}_b^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_{ab}^{\mathcal{A}} = 1]\right| > \mathsf{negl}(\lambda)$, then one can construct a PPT adversary $\mathcal{B}$ getting Key as input that breaks the key indistinguishability of BARG, where Key is either $\mathsf{BARG.Gen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|})$ or $\mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, 0)$ based on whether $\mathcal{A}$ is interacting with game $\mathsf{G}_b$ or $\mathsf{G}_{ab}$ respectively. The reduction then following in a similar manner as the SE key indistinguishability adversary described above.

**Lemma 5.** *Assuming somewhere soundness of* BARG*,*

$$\left|\Pr[\mathsf{G}_{ab}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_0^{\mathcal{A}} = 1]\right| \leq \mathsf{negl}(\lambda).$$

*Proof.* The only difference in Games $\mathsf{G}_{ab}$ and $\mathsf{G}_0$ is that there is an additional step which computes the true digest at index 0 and extracts at the $0^{th}$ index from $c_{\mathsf{even}}$ using the extraction function of SE. Finally, the adversary wins if and only if the extracted value matches the true digest along with the usual win conditions in the previous game.

Note that,

$$\begin{aligned}\left|\Pr[\mathsf{G}_{ab}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_0^{\mathcal{A}} = 1]\right| \leq \Pr[&\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \mathit{\Pi}) = 1 \wedge \\ &((x,y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_0 \neq \mathit{Step}(x, P, \mathsf{crs}, 0).\bar{\mathsf{h}}] \leq \\ &\Pr[\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \mathit{\Pi}) = 1 \wedge \mathsf{h}_0 \neq \mathit{Step}(x, P, \mathsf{crs}, 0).\bar{\mathsf{h}}].\end{aligned}$$

Let us assume that there exists a PPT adversary $\mathcal{A}$ such that for infinitely many values of $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \mathit{\Pi}) = 1 \wedge \mathsf{h}_0 \neq \mathit{Step}(x, y, P, \mathsf{crs}, 0).\bar{\mathsf{h}}] \geq \frac{1}{\mathsf{poly}(\lambda)}.$$

Notice that $\mathsf{h}_0 \neq \mathit{Step}(x, P, \mathsf{crs}, 0).\bar{\mathsf{h}}$ implies that at least one of the conditions $\mathsf{st}_0 = \mathsf{start}$, $H_x = \mathsf{rt}_0^1$, $H_P = \mathsf{rt}_0^2$ and $\mathsf{HT.Hash}(\mathsf{dk}, \square)$ having $\mathsf{rt}_0^3$ as root must not be true. If this is indeed true then our construction of $C_{\mathsf{index}}$ in Figure 3 implies that $(C_{\mathsf{index}}, 0) \notin \mathcal{L}^{\mathsf{index}}$.

We now construct the following PPT adversary $\mathcal{B}$ playing the semi-adaptive somewhere soundness game of the BARG as follows

Adversary $\mathcal{B}$ playing semi adaptive somewhere soundness game of BARG.
- $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_0})$
- $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.TGen}(1^\lambda, 1^M, I_{S_1})$
- $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.TGen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|}, 0)$
- $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
- $\mathsf{crs} \coloneqq (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk})$.
- $(P, \mathsf{aux}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{crs})$
- $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
- $((x,y), (c, \mathit{\Pi})) \leftarrow \mathcal{A}_2(\mathsf{crs}, P, \mathsf{aux})$
- return $(C_{\mathsf{index}}, \mathit{\Pi})$

By our assumption, it is clear that $\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1$ with non negligible probability but $(C_{\mathsf{index}}, 0) \notin \mathcal{L}^{\mathsf{index}}$. Thus, $\mathcal{B}$ will break the semi-adaptive somewhere soundness of $\mathsf{BARG}$ at index 0. Therefore, it must be the case that for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1 \wedge \mathsf{h}_0 \neq \mathit{Step}(x, y, P, \mathsf{crs}, 0).\bar{\mathsf{h}}] \leq \mathsf{negl}(\lambda)$$
$$\implies \left|\Pr[\mathsf{G}_{ab}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_0^{\mathcal{A}} = 1]\right| \leq \mathsf{negl}(\lambda)$$

.

Now, we transition from $\mathsf{G}_0$ to the base case for our induction, $\mathsf{G}_1$ using the following sequence of indistinguishable hybrids:

$\mathsf{G}_{0,a}$ Identical to $\mathsf{G}_0$ except we add an extraction: $(\mathsf{h}_1, \{b_1^k\}_{k\in[3]}, \{\Pi_1^k\}_{k\in[3]}, \Pi_i') \leftarrow \mathsf{SE.Ext_{odd}}(c_{\mathsf{odd}}, \mathsf{SE}.K_{\mathsf{odd}})$ which is not used in the hybrid, hence indistinguishability follows.

$\mathsf{G}_{0,b}$ The $\mathsf{BARG}$ key generation's trapdoor is changed from 0 to 1. This can be done due to key indistinguishability of $\mathsf{BARG}$.

$\mathsf{G}_{0,c}$ The winning condition is changed to: if $\mathsf{stSNARG.V}\left(\mathsf{crs}, (x, y), H_P, (c, \Pi)\right) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_0 = \mathit{Step}(x, y, P, \mathsf{crs}, 0).\bar{\mathsf{h}} \wedge \{b_1^k\}_{k\in[3]} = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{b} \wedge \{\mathsf{rt}_1^k\}_{k\in[3]} = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{rt}} \wedge \mathsf{st}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{st}}$, return 1

$\mathsf{G}_{0,d}$ The winning condition is changed to: if $\mathsf{stSNARG.V}\left(\mathsf{crs}, (x, y), H_P, (c, \Pi)\right) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_0 = \mathit{Step}(x, y, P, \mathsf{crs}, 0).\bar{\mathsf{h}} \wedge \{b_1^k\}_{k\in[3]} = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{b} \wedge \mathsf{rt}_1^3 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{rt}} \wedge \mathsf{st}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{st}} \wedge \mathsf{h}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{h}}$, return 1

$\mathsf{G}_{0,e}$ The winning condition is changed to: if $\mathsf{stSNARG.V}\left(\mathsf{crs}, (x, y), H_P, (c, \Pi)\right) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{h}}$, return 1

The indistinguishability of the last three hybrids follow from the following lemmas.

**Lemma 6.** *Assuming semi-adaptive somewhere soundness of* $\mathsf{BARG}$*, extraction correctness of* $\mathsf{SE}$*, read and write soundness of* $\mathsf{HT}$*,*

$$\left|\Pr[\mathsf{G}_{0,b}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_{0,c}^{\mathcal{A}} = 1]\right| \leq \mathsf{negl}(\lambda).$$

*Proof.* The only difference in Games $\mathsf{G}_{0,b}$ and $\mathsf{G}_{0,c}$ is that we have added some additional conditions for the adversary to win along with the ones in the previous game.

Note that,

$$\left|\Pr[\mathsf{G}_{0,b}^{\mathcal{A}} = 1] - \Pr[\mathsf{G}_{0,c}^{\mathcal{A}} = 1]\right| \leq \Pr[\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1 \wedge$$
$$\mathsf{h}_0 = \mathit{Step}(x, P, \mathsf{crs}, 0).\bar{\mathsf{h}} \wedge \Big(\{b_1^k\}_{k\in[3]} \neq \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{b}$$
$$\vee \{\mathsf{rt}_1^k\}_{k\in[3]} \neq \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{rt}} \vee \mathsf{st}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{st}}\Big)].$$

Let us assume that there exists a PPT adversary $\mathcal{A}$ such that for infinitely many values of $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{BARG.V}(\mathsf{BARG.crs}, C_{\mathsf{index}}, \Pi) = 1 \wedge \mathsf{h}_0 = \mathit{Step}(x, y, P, \mathsf{crs}, 0).\bar{\mathsf{h}} \\ \wedge \Big(\{b_1^k\}_{k\in[3]} \neq \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{b} \vee \{\mathsf{rt}_1^k\}_{k\in[3]} \neq \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{rt}} \\ \vee \mathsf{st}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{st}}\Big)] \geq \frac{1}{\mathsf{poly}(\lambda)}.$$

Notice that $\mathsf{h}_0 = \mathit{Step}(x, P, \mathsf{crs}, 0).\bar{\mathsf{h}}$ implies that the conditions $\mathsf{st}_0 = \mathsf{start}$, $H_x = \mathsf{rt}_0^1$, $H_P = \mathsf{rt}_0^2$ and $\mathsf{HT.Hash}(\mathsf{dk}, \Box)$ having $\mathsf{rt}_0^3$ as root are true. In other words, $\mathsf{h}_0$ is indeed the true digest at step 0.

Assuming extraction correctness of $\mathsf{SE}$, read and write soundness of $\mathsf{HT}$, we construct the following PPT adversary $\mathcal{B}$ playing the semi-adaptive somewhere soundness game of the $\mathsf{BARG}$ similar to the one in the proof of Lemma 5.

Thus, $\mathcal{B}$ will break the semi-adaptive somewhere soundness of $\mathsf{BARG}$ at index 1 if $(C_{\mathsf{index}}, 1) \notin \mathcal{L}^{\mathsf{index}}$.

It is now left to show that $(C_{\mathsf{index}}, 1) \notin \mathcal{L}^{\mathsf{index}}$.

Case 1 If the $\mathsf{SE}$ verifications in $C_{\mathsf{index}}$ do not all return 1, then by construction of $C_{\mathsf{index}}$, we have that $(C_{\mathsf{index}}, 1) \notin \mathcal{L}^{\mathsf{index}}$.

Case 2 All $\mathsf{SE}$ verifications return 1. Extraction Correctness/ Somewhere binding property of $\mathsf{SE}$ hash implies that $\mathsf{h}_0 = (\mathsf{st}_0, \mathsf{rt}_0^1, \mathsf{rt}_0^2, \mathsf{rt}_0^3), \mathsf{h}_1, \{b_1^k, \Pi_1^k\}_{k\in[3]}, \Pi_1'$ were indeed committed by the prover as the Turing machine output at step 0 and step 1. Now, let us analyze $\phi(\mathsf{h}_0, \mathsf{h}_1, \{b_1^k, \Pi_1^k\}_{k\in[3]}, \Pi_1')$. By assumption, we know that $\mathsf{h}_0 = \bar{\mathsf{h}_0}$, i.e., $\bar{\mathsf{st}}_0, \bar{\mathsf{rt}}_0^1, \bar{\mathsf{rt}}_0^2, \bar{\mathsf{rt}}_0^3 = \mathsf{st}_0, \mathsf{rt}_0^1, \mathsf{rt}_0^2, \mathsf{rt}_0^3$. $\mathsf{StepR}$ being a deterministic function ensures that $(l_1^1, l_1^2, l_1^3)$ are indeed the correct Turing machine memory locations to be read at step 1. Thus $(\bar{l}_1^1, \bar{l}_1^2, \bar{l}_1^3) = (l_1^1, l_1^2, l_1^3)$. This along with the deterministic nature of hash tree read write operations means that we must have,

- $(\bar{l}_1^1, \bar{l}_1^2, \bar{l}_1^3) \leftarrow \mathsf{StepR}(\bar{\mathsf{st}}_0)$
- $\Big\{(\bar{b}_1^j, \bar{\Pi}_1^k) := \mathsf{HT.Read}(\bar{\mathsf{tree}}_0^k, \bar{l}_1^k)\Big\}_{k\in[3]}$
- $(\bar{b}_1'^3, \bar{l}_1'^3, \bar{\mathsf{st}}_1) := \mathsf{StepW}(\bar{\mathsf{st}}_0, \bar{b}_1^1, \bar{b}_1^2, \bar{b}_1^3)$
- $(\bar{\mathsf{tree}}_1^3, \bar{\mathsf{rt}}_1^3, \bar{\Pi}_1') := \mathsf{HT.Write}(\bar{\mathsf{tree}}_0^3, \bar{l}_1'^3, \bar{b}_1'^3)$

Read and Write Completeness of the hash tree implies
$\mathsf{HT.VerRead}(\mathsf{dk}_1, \bar{\mathsf{rt}}_0^1, \bar{l}_1^1, \bar{b}_1^1, \bar{\Pi}_1^1) = 1$
$\mathsf{HT.VerRead}(\mathsf{dk}_2, \bar{\mathsf{rt}}_0^2, \bar{l}_1^2, \bar{b}_1^2, \bar{\Pi}_1^2) = 1$
$\mathsf{HT.VerRead}(\mathsf{dk}_3, \bar{\mathsf{rt}}_0^3, \bar{l}_1^3, \bar{b}_1^3, \bar{\Pi}_1^3) = 1$
$\mathsf{HT.VerWrite}(\mathsf{dk}_3, \bar{\mathsf{rt}}_0^3, \bar{l}_1'^3, \bar{b}_1'^3, \bar{\mathsf{rt}}_1^3 \bar{\Pi}_1') = 1$
If $\{b_1^k\}_{k\in[3]} \neq \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{b}$, then the read soundness assumption of $\mathsf{HT}$ implies that
$\big(\mathsf{HT.VerRead}(\mathsf{dk}, \bar{\mathsf{rt}}_0^1, \bar{l}_1^k, b_1^k, \Pi_1^k) = 1\big)_{k\in[3]}$ happens with a negligible probability. Thus, with all but negligible probability we have that $(C_{\mathsf{index}}, 1) \notin \mathcal{L}^{\mathsf{index}}$ and we are done.
Let us say this is not the case, i.e., $\{b_1^k\}_{k\in[3]} = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{b}$, then the deterministic nature of the Turing machine write function $\mathsf{StepW}$ implies that

$\mathsf{st}_1 = \bar{\mathsf{st}}_1$. Thus, for our assumption to be valid, it must be that $\{\mathsf{rt}_1^k\}_{k\in[3]} \neq \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{rt}}$. If $\mathsf{rt}_1^1 \neq \bar{\mathsf{rt}}_1^1 = \mathsf{rt}_0^1$ or $\mathsf{rt}_1^2 \neq \bar{\mathsf{rt}}_1^2 = \mathsf{rt}_0^2$, then the definition of $\phi$ implies that $(C_{\mathsf{index}}, 1) \notin \mathcal{L}^{\mathsf{index}}$. If this is not the case, then the only other possible option is $\mathsf{rt}_1^3 \neq \bar{\mathsf{rt}}_1^3$. Now, the write soundness of $\mathsf{HT}$ implies that with all but negligible probability, $\mathsf{HT.VerWrite}(\mathsf{dk}_3, \bar{\mathsf{rt}}_0^3, \bar{l}_1'^3, \bar{b}_1'^3, \mathsf{rt}_1^3, \Pi_1) \neq 1$ must hold. If this is indeed true then our construction of $C_{\mathsf{index}}$ in Figure 3 implies that $(C_{\mathsf{index}}, 1) \notin \mathcal{L}^{\mathsf{index}}$.

**Lemma 7.**
$$\Pr[\mathsf{G}_{0,c}^{\mathcal{A}} = 1] = \Pr[\mathsf{G}_{0,d}^{\mathcal{A}} = 1].$$

*Proof.* Note that by definition, $\mathsf{h}_1 = \mathsf{st}_1, \mathsf{rt}_1^1, \mathsf{rt}_1^2, \mathsf{rt}_1^3$. We already have that $\mathsf{rt}_1^1, \mathsf{rt}_1^2, \mathsf{rt}_1^3 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{rt}}$ and $\mathsf{st}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{st}}$. Thus $\mathsf{h}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{h}}$ if and only if $\mathsf{rt}_1^3 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{rt}} \wedge \mathsf{st}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{st}}$.

**Lemma 8.**
$$\Pr[\mathsf{G}_{0,d}^{\mathcal{A}} = 1] \leq \Pr[\mathsf{G}_{0,e}^{\mathcal{A}} = 1].$$

*Proof.* The number of conditions for the adversary to win simply decreases from Game $\mathsf{G}_{0,d}$ to Game $\mathsf{G}_{0,e}$, thus the probability of success must not increase.

A closer observation shows that $\mathsf{G}_{0,e}$ is indeed identical to the case when one puts $i = 1$ in game $\mathsf{G}_i$.

Combining these together, we show the base case of the induction to be true. Thus,

$$\Pr[\mathsf{G}^{\mathcal{A}} = 1] \leq \Pr[\mathsf{G}_1^{\mathcal{A}} = 1] + \mathsf{negl}(\lambda).$$

Assuming that our induction hypothesis holds for some $j \in [T-1]$, we prove that it holds for $j+1$ as well. We note that by chain rule, it suffices to show that $\Pr[\mathsf{G}_j^{\mathcal{A}} = 1] \leq \Pr[\mathsf{G}_{j+1}^{\mathcal{A}} = 1] + \mathsf{negl}(\lambda)$. We can show this by a sequence of indistinguishable inner hybrids to transition from Game $\mathsf{G}_j$ to $\mathsf{G}_{j+1}$ which look like the following:

- $\mathsf{G}_{j,a}$ Identical to $\mathsf{G}_j$ except the $\mathsf{SE}$ hash extraction is done at $S_{j+1}$ instead of $S_j$. Indistinguishability follows from the key indistinguishability of $\mathsf{SE}$ hash.
- $\mathsf{G}_{j,b}$ Extraction for one of the $\mathsf{SE}$ hashes changes from $I_{S_{j-1}}$ to $I_{S_{j+1}}$. However, this does not affect the reduction in any way as extraction at indices $j-1$ and $j+1$ are not used by the reduction at any stage.
- $\mathsf{G}_{j,c}$ The $\mathsf{BARG}$ key generation has a trapdoor at $j+1$. This can be done due to key indistinguishability of $\mathsf{BARG}$.
- $\mathsf{G}_{j,d}$ The winning condition is changed to: if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_j = \mathit{Step}(x, y, P, \mathsf{crs}, j).\bar{\mathsf{h}}_j \wedge \{b_{j+1}^k\}_{k\in[3]} = \mathit{Step}(x, y, P, \mathsf{crs}, j+1).\bar{b}_{j+1} \wedge \mathsf{rt}_{j+1}^3 = \mathit{Step}(x, y, P, \mathsf{crs}, j+1).\bar{\mathsf{rt}}_{j+1} \wedge \mathsf{st}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{st}}$, return 1.
- $\mathsf{G}_{j,e}$ The winning condition is changed to: if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_j = \mathit{Step}(x, y, P, \mathsf{crs}, j).\bar{\mathsf{h}}_j \wedge \{b_{j+1}^k\}_{k\in[3]} = \mathit{Step}(x, y, P, \mathsf{crs}, j+1).\bar{b}_{j+1} \wedge \mathsf{rt}_{j+1}^3 = \mathit{Step}(x, P, \mathsf{crs}, j+1).\bar{\mathsf{rt}}_{j+1} \wedge \mathsf{st}_1 = \mathit{Step}(x, y, P, \mathsf{crs}, 1).\bar{\mathsf{st}} \wedge \mathsf{h}_{j+1} = \mathit{Step}(x, y, P, \mathsf{crs}, j+1).\bar{\mathsf{h}}$, return 1

$\mathsf{G}_{j,f}$ if $\mathsf{stSNARG.V}(\mathsf{crs}, (x, y), H_P, (c, \Pi)) = 1 \wedge ((x, y), T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}} \wedge \mathsf{h}_{j+1} = \mathit{Step}(x, y, P, \mathsf{crs}, j + 1).\bar{\mathsf{h}}$, return 1

The last three steps follow identically as Lemmas 67,8.

Observe $\mathsf{G}_{j,f}$ is identical to outer Game $\mathsf{G}_{j+1}$ with indices renamed.

Thus, combining the lemmas above, we get

**Lemma 9.** *Assuming extraction correctness of* SE*, semi-adaptive somewhere soundness of* BARG*, read and write soundness of* HT*,*

$$\Pr[\mathsf{G}_j^{\mathcal{A}} = 1] \leq \Pr[\mathsf{G}_{j+1}^{\mathcal{A}} = 1] + \mathsf{negl}(\lambda).$$

This follows from the combination of previous lemmas where we showed that the winning probability in the sequence of inner hybrids are either negligibly close to each other or increases (from Game $\mathsf{G}_{j,e}$ to Game $\mathsf{G}_{j,f}$).

Finally, we will show that the winning probability of $\mathcal{A}$ is 0 in the final game $\mathsf{G}_T$.

**Lemma 10.** *Assuming extraction correctness of* SE *hash,*

$$\Pr[\mathsf{G}_T^{\mathcal{A}} = 1] = 0.$$

*Proof.* The extraction correctness of SE ensures that $\mathsf{h}_T$ was indeed the state committed by the prover. Now, $\mathsf{h}_T = \bar{\mathsf{h}}_T$ cannot be true since our assumption of $(x, y, T, P, H_P, \mathsf{crs}) \notin \mathcal{L}_{\mathcal{TM}}$ means that Turing Machine state after $T$ steps cannot be an accept state. Thus, the adversary's win conditions cannot be simultaneously satisfied.

Note that this step does not require us to resort to BARG soundness. Due to our specific construction of $\bar{\mathsf{h}}_T$, all we need ensure is that the state committed by the prover does not correspond to the correct state.

Compiling the lemmas together and using chain rule, it must be true that

$$\Pr[\mathsf{G}^{\mathcal{A}} = 1] \leq \mathsf{negl}(\lambda)$$

which is a contradiction to our assumption that the scheme is not sound.

**Lemma 11.** *Assuming $T = \mathsf{poly}(m, n)$, $T, m, n \leq 2^\lambda$, the* stSNARG *protocol in Figure 2 implies the unconditional existence of a publicly verifiable non interactive succinct delegation scheme* sDel *as defined above.*

*Proof.* We provide an explicit construction of sDel assuming a semi-trusted SNARG stSNARG. Without loss of generality, we can assume that $T$ is known a-priory.

- $\mathsf{sDel.Setup}(1^\lambda)$: Run $\mathsf{stSNARG.Setup}$ to generate $\mathsf{crs}$.
- $\mathsf{sDel.ProgAuth}(1^\lambda, \mathsf{crs})$: Generate a program $P \in \{0, 1\}^m$, state and run $\mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$ to get $H_P$.
- $\mathsf{sDel}.I(1^\lambda, \mathsf{crs})$: Generate $x \in \{0, 1\}^n$ .

– $\mathsf{sDel}.W(\mathsf{crs}, P, \mathsf{state}, H_P, x)$: Generate $y \in \{0,1\}$ and run $\mathsf{stSNARG.P}(\mathsf{crs}, P, x, y, H_P)$ to get $\Pi$.
– $\mathsf{sDel}.V(\mathsf{crs}, x, y, H_P, \Pi)$: Run $\mathsf{stSNARG.V}(\mathsf{crs}, x, y, H_P, \Pi)$ return V's output.

Completeness and soundness of sDel follows from the completeness of stSNARG in a straightforward way. Refer to Supplementary material for detailed analysis. The proof size and verifier run time of stSNARG is $\mathsf{poly}(\lambda, \log T)$=$\mathsf{poly}(\lambda, \log |P|, \log |x|)$. Similarly, the prover run time of sDel is also $\mathsf{poly}(\lambda, |P|, |x|)$.

## 6 Semi-Trusted Succinct Non-Interactive Argument with Zero Knowledge (ZK-stSNARG)

A publicly verifiable semi-trusted non interactive argument with zero-knowledge scheme
ZKstSNARG : (ZKstSNARG.Setup, ZKstSNARG.TrustHash, ZKstSNARG.P, ZKstSNARG.V) is defined as

– $\mathsf{ZKstSNARG.Setup}(1^\lambda, 1^T)$: A randomized setup algorithm which on input security parameter $\lambda$, and number of Turing Machine steps $T$, outputs crs.
– $\mathsf{ZKstSNARG.TrustHash}(\mathsf{crs}, P)$: A deterministic an honest algorithm which on input crs and a program $P \in \{0,1\}^m$ for some $m < 2^\lambda$, computes a succinct digest $H_P$ of $P$. It then produces a statistically binding and extractable commitment $C_P$ of $H_P$ under randomness $r_1$. It then gives out a pair public output $\mathsf{POut} = C_P$ and private output $\mathsf{SOut} = (H_P, r)$. Here SOut is made available to the prover only.
– $\mathsf{ZKstSNARG.P}(\mathsf{crs}, P, x, y, \mathsf{SOut}, \mathsf{POut})$: A deterministic prover algorithm which on input the crs, $P \in \{0,1\}^m$ for some $m < 2^\lambda$, $x \in \{0,1\}^n$ for some $n < 2^\lambda$, $y \in \{0,1\}$, SOut, and POut outputs a proof $\Pi$.
– $\mathsf{ZKstSNARG.V}(\mathsf{crs}, x, y, \mathsf{POut}, \Pi)$: A deterministic verification algorithm which on input crs, $x$, $y$, public output POut of stSNARG.TrustHash and proof $\Pi$, either accepts(output 1) or rejects(output 0) it.

We define the following language

$$\mathcal{L}_{\mathcal{TM}} \coloneqq \{(P, x, y, T, \mathsf{POut}, \mathsf{crs}) \,\big|\, \exists (H_P, r_1) \text{ such that } \mathcal{TM}(P, x, y) = 1 \wedge \\ (\mathsf{POut}, (H_P, r_1)) = \mathsf{ZKstSNARG.TrustHash}(\mathsf{crs}, P)\}.$$

A ZKstSNARG satisfies the standard completeness, soundness and efficiency properties as stSNARG. It also has an additional property:

**Non Interactive Zero Knowledge.** For all $(P, x, y, T, \mathsf{POut}, \mathsf{crs}) \in \mathcal{L}_{\mathcal{TM}}$, there exists a PPT simulator $\mathsf{Sim} \coloneqq (\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3)$ such that the distributions of

$$(\mathsf{crs}, x, y, \mathsf{POut}, \Pi) \big| (\mathsf{crs}, \mathsf{aux}) \leftarrow \mathsf{Sim}_1(1^\lambda, 1^T), \\ (\mathsf{POut}, \mathsf{aux}') \leftarrow \mathsf{Sim}_2(\mathsf{crs}, \mathsf{aux}), \\ \Pi \leftarrow \mathsf{Sim}_3(\mathsf{aux}', \mathsf{crs}, (x, y), \mathsf{POut})$$

and

$$(\mathsf{crs}, x, y, \mathsf{POut}, \mathit{\Pi}) \big| \mathsf{crs} \leftarrow \mathsf{ZKstSNARG.Setup}(1^\lambda, 1^T), \\ (\mathsf{POut}, \mathsf{SOut}) \leftarrow \mathsf{ZKstSNARG.TrustHash}(\mathsf{crs}, P), \\ \mathit{\Pi} \leftarrow \mathsf{ZKstSNARG.P}(\mathsf{crs}, P, x, y, \mathsf{POut}, \mathsf{SOut})$$

are indistinguishable.

To achieve non interactive zero knowledge, we use the following additional primitives, namely (1) a statistically binding extractable commitment scheme $\mathsf{Com}_{\mathsf{bind}}$ as defined in Section 3, and (2) a Non Interactive Zero Knowledge argument $\mathsf{NIZK} := (\mathsf{NIZK.Gen}, \mathsf{NIZK.P}, \mathsf{NIZK.V})$.

The protocol in Figure 5 demonstrates the extension of $\mathsf{stSNARG}$ to achieve Zero-Knowledge. The CRS in Figure 5 contains a statistically binding commitment to 0. This lets us extend $\mathcal{L}_{\mathcal{TM}}$ to the language,

$$\mathcal{L}_{\mathsf{hyb}} := \Bigg\{ (P, x, y, T, C_P, \mathsf{crs}) \Big| \exists (H_P, r_1) \text{ such that } \mathcal{TM}(P, x, y) = 1 \\ \wedge (C_P, (H_P, r_1)) = \mathsf{ZKstSNARG.TrustHash}(\mathsf{crs}, P) \\ \vee \Big( \exists r \text{ such that } \mathsf{crs} \text{ contains a commitment to 1 under randomness } r \Big). \Bigg\}$$

such that any witness to $\mathcal{L}_{\mathcal{TM}}$ is vacuously a witness to $\mathcal{L}_{\mathsf{hyb}}$ due to binding property of the commitment. We use $\mathsf{NIZK}$ for the following $NP$ language:

$$\mathcal{L} := \Bigg\{ (c.\mathsf{com}, \mathit{\Pi}.\mathsf{com}, (\mathsf{crs}, x, y, T), C_P) \Big| \exists r_1, r_2, r_3, r_4, c, \mathit{\Pi}, H_P \text{ such that} \\ \Bigg( C_P = \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_1, H_P; r_1) \wedge c.\mathsf{com} = \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_2, c; r_2) \\ \wedge \mathit{\Pi}.\mathsf{com} = \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_3, \mathit{\Pi}; r_3) \wedge \mathsf{stSNARG.V}(\mathsf{crs}, ((x, y), T, H_P), (c, \mathit{\Pi})) = 1 \Bigg) \\ \vee \mathsf{crs} \text{ contains } \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_4, 1; r_4) \Bigg\}$$

Also, note that in this construction, the underlying $\mathsf{stSNARG}$ is built for the index circuit $C'_{\mathsf{index}}$ which is identical to $C_{\mathsf{index}}$ except that $H_P$ is a part of the input and not hard-coded in the circuit as it is not known to the verifier.

**Theorem 4.** *Assuming the existence of semi-trusted SNARGs and Extractable Statistically Binding Commitment Schemes, and NIZK as described in sections 3 and 5, Figure 5 is a publicly verifiable non-interactive semi-trusted SNARG with zero knowledge such that CRS size, proof size and verifier time are* $\mathsf{poly}(\lambda, \log T)$ *and prover run time is* $\mathsf{poly}(\lambda, T)$.

**Protocol 2 (stSNARG with Zero-Knowledge)**

```
– $\mathsf{ZKstSNARG.Setup}(1^\lambda, T)$ :
    • $\mathsf{crs}_1 \leftarrow \mathsf{stSNARG.Setup}(1^\lambda, 1^T)$
    • $\mathsf{Com_{bind}}.Key_1 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
    • $\mathsf{Com_{bind}}.Key_2 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
    • $\mathsf{Com_{bind}}.Key_3 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
    • $\mathsf{Com_{bind}}.Key_4 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
    • $r_4 \leftarrow_\$ \{0,1\}^\lambda, z \leftarrow \mathsf{Com.C}(\mathsf{Com}.Key_4, 0; r_4)$
    • $\mathsf{NIZK.crs} \leftarrow \mathsf{NIZK.Gen}(1^\lambda)$
    • $\mathit{return}\ (\mathsf{crs}_1, \mathsf{Com_{bind}}.Key_1.\mathsf{Com_{bind}}.Key_2, \mathsf{Com_{bind}}.Key_3, z, \mathsf{NIZK.crs})$.
– $\mathsf{ZKstSNARG.TrustHash}(\mathsf{crs}, P)$
    • $H_P \leftarrow \mathsf{stSNARG.TrustHash}(\mathsf{crs}, P)$
    • $r_1 \leftarrow_\$ \{0,1\}^\lambda,\ C_P \leftarrow \mathsf{Com.C}(\mathsf{Com_{bind}}.Key_1, H_P; r_1)\ \mathit{return}\ (\mathsf{SOut} :=
      (P, r_1), \mathsf{POut} := C_P)$.
– $\mathsf{ZKstSNARG.P}(\mathsf{crs}, x, y, \mathsf{SOut}, \mathsf{POut})$ :
    • $(c, \Pi) \leftarrow \mathsf{stSNARG.P}(\mathsf{crs}, x, y, H_P)$
    • $r_2 \leftarrow_\$ \{0,1\}^\lambda, c.\mathsf{com} \leftarrow \mathsf{Com.C}(\mathsf{Com_{bind}}.Key_2, c; r_2)$
    • $r_3 \leftarrow_\$ \{0,1\}^\lambda, \Pi.\mathsf{com} \leftarrow \mathsf{Com.C}(\mathsf{Com_{bind}}.Key_3, \Pi; r_3)$
    • $\mathsf{NIZK}.\Pi \leftarrow \mathsf{NIZK.Prove}\Big(\mathsf{NIZK.crs}, (c.\mathsf{com}, \Pi.\mathsf{com}, (\mathsf{crs}, x, y, T), C_P)$ ,
      $((H_P, r_1), (c, r_2), (\Pi, r_3), \perp)\Big)$
    • $\mathit{return}\ (c.\mathsf{com}, \Pi.\mathsf{com}, \mathsf{NIZK}.\Pi)$.
– $\mathsf{ZKstSNARG.V}(\mathsf{crs}, (x, y), \mathsf{POut} = C_P, c.\mathsf{com}, \Pi.\mathsf{com}, \mathsf{NIZK}.\Pi)$ :
    • $\mathit{return}\ 1\ \mathit{if\ and\ only\ if}$
      $\mathsf{NIZK.V}(\mathsf{NIZK.crs}, (c.\mathsf{com}, \Pi.\mathsf{com}, (\mathsf{crs}, x, y, T), C_P), \mathsf{NIZK}.\Pi) = 1$.
```

Fig. 5: Semi-Trusted Universal Turing Machine Delegation with Non Interactive Zero-Knowledge

*Completeness and Efficiency.* Completeness follows from the completeness of the underlying stSNARG, NIZK and the binding property of the commitment. Similarly succinctness follows from the efficiency of stSNARG, NIZK, and the binding commitment $\mathsf{Com_{bind}}$.

*Soundness.* The soundness following by a straightforward reduction using the CRS indistinguishability and Statistical Binding of $\mathsf{Com_{bind}}$, and the soundness of the underlying stSNARG. We can construct an adversary that breaks the soundness of the underlying stSNARG using the following steps:

1. Change the keys for $\mathsf{Com_{bind}}$ to be generated by TGen. This can be done due to CRS indistinguishability.
2. The reduction can now extract the committed proof from $c.\mathsf{com}$ and $\Pi.\mathsf{com}$. This is because the reduction has access to the trapdoor commitment key.
3. The stSNARG can now output the extracted proof. The extraction correctness of $\mathsf{Com_{bind}}$ ensures that if ZKstSNARG is not sound, then this adversary breaks the soundness of stSNARG.

A formal analysis is presented in the Supplementary Material.

*Zero-Knowledge.*

```
$zk$ − stSNARG Simulator $\mathsf{NIZK.Sim} \coloneqq (\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3)$
• $\mathsf{Sim}_1(1^\lambda, 1^T)$ :
  1. $\mathsf{SE}.K_{\mathsf{even}} \leftarrow \mathsf{SE.Gen}(1^\lambda, 1^{M_{\lambda,T}}, 1^{L_\lambda})$
  2. $\mathsf{SE}.K_{\mathsf{odd}} \leftarrow \mathsf{SE.Gen}(1^\lambda, 1^M, 1^L)$
  3. $\mathsf{BARG.crs} \leftarrow \mathsf{BARG.Gen}(1^\lambda, 1^{T+1}, 1^{|C_{\mathsf{index}}|})$
  4. $\mathsf{dk} \leftarrow \mathsf{HT.Gen}(1^\lambda)$
  5. $\mathsf{Com}_{\mathsf{bind}}.Key_1 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
  6. $\mathsf{Com}_{\mathsf{bind}}.Key_2 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
  7. $\mathsf{Com}_{\mathsf{bind}}.Key_3 \leftarrow \mathsf{Com.Gen}(1^\lambda)$
  8. $\mathsf{Com}_{\mathsf{bind}}.Key_4 \leftarrow \mathsf{Com.Gen}(1^\lambda), r_4 \leftarrow_\$ \mathbb{N}, z \leftarrow \mathsf{Com.C}(\mathsf{Com}.Key_4, 1; r_4)$
  9. $\mathsf{NIZK.crs} \leftarrow \mathsf{NIZK.Gen}(1^\lambda)$
  10. return
      $\mathsf{crs} \coloneqq (\mathsf{SE}.K_{\mathsf{even}}, \mathsf{SE}.K_{\mathsf{odd}}, \mathsf{BARG.crs}, \mathsf{dk}, \mathsf{Com}_{\mathsf{bind}}.Key_1.\mathsf{Com}_{\mathsf{bind}}.Key_2, \mathsf{Com}_{\mathsf{bind}}.Key_3, z, \mathsf{NIZK.crs})$
      and $\mathsf{aux} \coloneqq r_4$
• $\mathsf{Sim}_2(\mathsf{crs}, \mathsf{aux})$ :
  1. $r_1 \leftarrow_\$ \{0,1\}^\lambda$, $C_P \leftarrow \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_1, 0; r_1)$ return $\mathsf{POut} \coloneqq C_P$.
  2. return $(\mathsf{POut}, \mathsf{aux}' \coloneqq \mathsf{aux})$
• $\mathsf{Sim}_3(\mathsf{crs}, \mathsf{aux}', (x, y), \mathsf{POut} \coloneqq C_P)$ :
  1. $r_2 \leftarrow_\$ \{0,1\}^\lambda, c.\mathsf{com} \leftarrow \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_2, 0; r_2)$
  2. Generate a dummy proof $\hat{\Pi}$
  3. $r_3 \leftarrow_\$ \{0,1\}^\lambda, \Pi.\mathsf{com} \leftarrow \mathsf{Com.C}(\mathsf{Com}_{\mathsf{bind}}.Key_3, 0; r_3)$
  4. $\mathsf{NIZK}.\Pi \leftarrow \mathsf{NIZK.Prove}\left(\mathsf{NIZK.crs}, (c.\mathsf{com}, \Pi.\mathsf{com}, (\mathsf{crs}, x, y, T), C_P), (\bot, \bot, \bot, \mathsf{aux})\right)$
  5. return $(c.\mathsf{com}, \Pi.\mathsf{com}, \mathsf{NIZK}.\Pi)$.
```

The proof of zero-knowledge follows from a sequence of hybrids.

– We define a game $\mathsf{G}'$ which is identical to $\mathsf{G}_0$ except that $\mathsf{crs}$ has a commitment of 1 instead of 0. Note that an honest prover does not make use of this section of the $\mathsf{crs}$ in its proof. Consider $\mathsf{hyb}'$ as the output distribution of intermediate $\mathsf{G}'$. All other algorithms in $\mathsf{G}'$ remains identical as $\mathsf{G}_0$. $\mathsf{hyb}_0$ must be indistinguishable from $\mathsf{hyb}'$, otherwise we can construct an efficient adversary that breaks the computational hiding property of $\mathsf{Com}_{\mathsf{bind}}$.
– The hybrid game $\mathsf{G}''$ with output distribution $\mathsf{hyb}''$ works like $\mathsf{G}'$ except $\mathsf{stSNARG.P}$ computes $(c.\mathsf{com}, \mathsf{NIZK}.\Pi)$ honestly and then ignores $c.\mathsf{com}$ and outputs $(c_1, \mathsf{NIZK}.\Pi)$ where $c_1$ is the statistical binding commitment to the 0 string using $\mathsf{Com}_{\mathsf{bind}}$. The indistinguishability of $\mathsf{hyb}'$ and $\mathsf{hyb}''$ follows from the computational hiding property of $\mathsf{Com}_{\mathsf{bind}}$.
– We now define another hybrid game $\mathsf{G}'''$ where everything remains identical as $\mathsf{G}''$ but the NIZK proof $\mathsf{NIZK.P}$ proves that $\mathsf{crs}$ has a commitment of 1 using randomness $r$ as a witness. This is indeed a valid witness for the same language $\mathcal{L}^*_{\mathsf{hyb}}$. Observe that $\mathsf{G}''$ and $\mathsf{G}'''$ have identical CRS. However, $\mathsf{NIZK.P}$ in each case uses different witnesses, namely $r$ and $((c, r_{\mathsf{com}_2}), \Pi)$ respectively. Thus, the Witness Indistinguishability of NIZK implies indistinguishability of $\mathsf{G}''$ and $\mathsf{G}'''$.
– In the next hybrid $\mathsf{G}''''$, trusted commitment generator is replaced by $\mathsf{Sim}_2$ which on input $\mathsf{crs}$ simply outputs a hiding commitment to the 0 string. Note that the output of $\mathsf{Sim}_2$ is not used anywhere else in the proof and its output is computationally indistinguishable from the public output of $\mathsf{ZKstSNARG.TrustHash}(\mathsf{crs}, P)$ because of the hiding property of commitment scheme.

– In the final game $\mathsf{G}_1$, $\mathsf{Sim}_1$ uses the same $\mathsf{crs}$ as the previous hybrid. $\mathsf{Sim}_3$ ignores all operations performed by the prover and only outputs $c_1$ which is the statistical binding commitment to the 0 string using $\mathsf{Com}_{\mathsf{bind}}$ and sends a NIZK proof as $\mathsf{G}''''$. The output distributions of $\mathsf{G}''''$ and $\mathsf{G}_1$ are indeed identical as the output of $\mathsf{Sim}_3$ solely depends on the output of $\mathsf{Sim}_1, \mathsf{Sim}_2$ and the commitment of the 0 string $c_1$.

Combining all the hybrids, we prove that $\mathsf{G}_0$ and $\mathsf{G}_1$ have output distributions which are computationally indistinguishable.

*Public Verifiable Non Interactive Succinct Delegation with Zero Knowledge* A direct extension of Lemma 11 gives us the following corollary,

**Corollary 2.** *Assuming $T = \mathsf{poly}(m, n)$, $T, m, n \leq 2^\lambda$, the* ZKstSNARG *protocol in Figure 5 implies the unconditional existence of a publicly verifiable non interactive succinct delegation scheme with zero knowledge.*

The zero knowledge simulator for the delegation scheme
$\mathsf{zk} - \mathsf{sDel.Sim} := (\mathsf{zk} - \mathsf{sDel.Sim}_1, \mathsf{zk} - \mathsf{sDel.Sim}_2)$ can simply run the stSNARG ZK-simulator. More specifically, $\mathsf{zk} - \mathsf{sDel.Sim}_1$ and $\mathsf{zk} - \mathsf{sDel.Sim}_2$ call $zk - \mathsf{stSNARG.Sim}_1$ and $zk - \mathsf{stSNARG.Sim}_2$ respectively above. The proof follows in a straightforward manner, hence we skip the details.

## 7 Acknowledgements

This research was supported in part from a Simons Investigator Award, DARPA SIEVE award, NTT Research, BSF grant 2012378, a Xerox Faculty Research Award, a Google Faculty Research Award, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through Award HR00112020024.

## References

1. Badrinarayanan, S., Kalai, Y.T., Khurana, D., Sahai, A., Wichs, D.: Succinct delegation for low-space non-deterministic computation. In: Diakonikolas, I., Kempe, D., Henzinger, M. (eds.) 50th ACM STOC. pp. 709–721. ACM Press (Jun 2018)
2. Bitansky, N., Canetti, R., Chiesa, A., Goldwasser, S., Lin, H., Rubinstein, A., Tromer, E.: The hunting of the snark. Journal of Cryptology 30(4), 989–1066 (2017)
3. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKS and proof-carrying data. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC. pp. 111–120. ACM Press (Jun 2013)
4. Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., Paneth, O.: Succinct non-interactive arguments via linear interactive proofs. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 315–333. Springer, Heidelberg (Mar 2013)
5. Bitansky, N., Kalai, Y.T., Paneth, O.: Multi-collision resistance: a paradigm for keyless hash functions. In: Diakonikolas, I., Kempe, D., Henzinger, M. (eds.) 50th ACM STOC. pp. 671–684. ACM Press (Jun 2018)
6. Brakerski, Z., Holmgren, J., Kalai, Y.T.: Non-interactive delegation and batch NP verification from standard computational assumptions. In: Hatami, H., McKenzie, P., King, V. (eds.) 49th ACM STOC. pp. 474–482. ACM Press (Jun 2017)

7. Brakerski, Z., Kalai, Y.: Witness indistinguishability for any single-round argument with applications to access control. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part II. LNCS, vol. 12111, pp. 97–123. Springer, Heidelberg (May 2020)
8. Canetti, R., Chen, Y., Holmgren, J., Lombardi, A., Rothblum, G.N., Rothblum, R.D., Wichs, D.: Fiat-Shamir: from practice to theory. In: Charikar, M., Cohen, E. (eds.) 51st ACM STOC. pp. 1082–1090. ACM Press (Jun 2019)
9. Choudhuri, A.R., Jain, A., Jin, Z.: Non-interactive batch arguments for NP from standard assumptions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 394–423. Springer, Heidelberg, Virtual Event (Aug 2021)
10. Choudhuri, A.R., Jain, A., Jin, Z.: Snargs for $\mathcal{P}$ from *lwe*. Cryptology ePrint Archive (2021)
11. Chung, K.M., Kalai, Y., Vadhan, S.P.: Improved delegation of computation using fully homomorphic encryption. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 483–501. Springer, Heidelberg (Aug 2010)
12. Cormode, G., Mitzenmacher, M., Thaler, J.: Practical verified computation with streaming interactive proofs. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. pp. 90–112 (2012)
13. Damgård, I., Faust, S., Hazay, C.: Secure two-party computation with low communication. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 54–74. Springer, Heidelberg (Mar 2012)
14. Feige, U., Lapidot, D., Shamir, A.: Multiple non-interactive zero knowledge proofs based on a single random string. In: Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science. pp. 308–317. IEEE (1990)
15. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (Aug 2010)
16. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (May 2013)
17. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. Journal of the ACM (JACM) 62(4), 1–64 (2015)
18. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340. Springer, Heidelberg (Dec 2010)
19. Holmgren, J., Lombardi, A., Rothblum, R.D.: Fiat–shamir via list-recoverable codes (or: parallel repetition of gmw is not zero-knowledge). In: Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing. pp. 750–760 (2021)
20. Hubacek, P., Wichs, D.: On the communication complexity of secure function evaluation with long output. In: Roughgarden, T. (ed.) ITCS 2015. pp. 163–172. ACM (Jan 2015)
21. Kalai, Y., Paneth, O., Yang, L.: On publicly verifiable delegation from standard assumptions. Cryptology ePrint Archive (2018)
22. Kalai, Y.T., Paneth, O.: Delegating RAM computations. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B, Part II. LNCS, vol. 9986, pp. 91–118. Springer, Heidelberg (Oct / Nov 2016)
23. Kalai, Y.T., Paneth, O., Yang, L.: How to delegate computations publicly. In: Charikar, M., Cohen, E. (eds.) 51st ACM STOC. pp. 1115–1124. ACM Press (Jun 2019)

24. Kalai, Y.T., Raz, R., Rothblum, R.D.: Delegation for bounded space. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC. pp. 565–574. ACM Press (Jun 2013)
25. Kalai, Y.T., Raz, R., Rothblum, R.D.: How to delegate computations: the power of no-signaling proofs. In: Shmoys, D.B. (ed.) 46th ACM STOC. pp. 485–494. ACM Press (May / Jun 2014)
26. Kalai, Y.T., Vaikuntanathan, V., Zhang, R.Y.: Somewhere statistical soundness, post-quantum security, and snargs. In: Theory of Cryptography Conference. pp. 330–368. Springer (2021)
27. Kilian, J.: A note on efficient zero-knowledge proofs and arguments. In: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. pp. 723–732 (1992)
28. Lipmaa, H.: Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 169–189. Springer, Heidelberg (Mar 2012)
29. Lombardi, A., Schaeffer, L.: A note on key agreement and non-interactive commitments. Cryptology ePrint Archive (2019)
30. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) CRYPTO'87. LNCS, vol. 293, pp. 369–378. Springer, Heidelberg (Aug 1988)
31. Micali, S.: Computationally sound proofs. SIAM Journal on Computing 30(4), 1253–1298 (2000)
32. Paneth, O., Rothblum, G.N.: On zero-testable homomorphic encryption and publicly verifiable non-interactive arguments. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part II. LNCS, vol. 10678, pp. 283–315. Springer, Heidelberg (Nov 2017)
33. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252. IEEE (2013)
34. Parno, B., Raykova, M., Vaikuntanathan, V.: How to delegate and verify in public: Verifiable computation from attribute-based encryption. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 422–439. Springer, Heidelberg (Mar 2012)
35. Peikert, C., Shiehian, S.: Noninteractive zero knowledge for NP from (plain) learning with errors. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 89–114. Springer, Heidelberg (Aug 2019)
36. Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. SIAM Journal on Computing 50(3), STOC16–255 (2019)
37. Setty, S., Vu, V., Panpalia, N., Braun, B., Blumberg, A.J., Walfish, M.: Taking {Proof-Based} verified computation a few steps closer to practicality. In: 21st USENIX Security Symposium (USENIX Security 12). pp. 253–268 (2012)
38. Setty, S.T., McPherson, R., Blumberg, A.J., Walfish, M.: Making argument systems for outsourced computation practical (sometimes). In: NDSS. vol. 1, p. 17 (2012)
39. Tauman Kalai, Y., Raz, R., Rothblum, R.D.: Delegation for bounded space. In: Proceedings of the forty-fifth annual ACM symposium on Theory of computing. pp. 565–574 (2013)
40. Thaler, J., Roberts, M., Mitzenmacher, M., Pfister, H.: {Verifiable} computation with massively parallel interactive proofs. In: 4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 12) (2012)